NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INTERACTIVE COMPUTER AIDED DESIGN
AND ANALYSIS PACKAGE

by

John Curtis Bordeaux

September 1986

Thesis Advisor: Larry W. Abbott

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable) 62	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
AN INTERACTIVE COMPUTER AIDED DESIGN AND ANALYSIS PACKAGE

12 PERSONAL AUTHOR(S)
John Curtis Bordeaux

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	86 September 26	172

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	SURE MARKOV

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

In this thesis a NASA developed Markov reliability analysis tool will be ported to a relatively inexpensive IBM-AT environment. Currently the Markov analysis took, called SURE, is not widely utilized because it runs in an expensive environment consisting of a VAX, megatex display, and template graphics software. Although substantial savings can be made by running SURE without the expensive graphics, the user friendliness of the tool is dramatically degraded by the lack of graphics. Accordingly a C program using the inexpensive GEM graphics environment has been written. The program is user friendly; it uses menus for option selections and prompts for data entry.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Prof Larry Abbott	22b TELEPHONE (Include Area Code) (408)646-2379; 22c. OFFICE SYMBOL 62At

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

Approved for public release; distribution is unlimited.

An Interactive Computer Aided Design and Analysis Package

by

John C. Bordeaux
Major, United States Marine Corps
B.S., Old Dominion University, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

ABSTRACT

This thesis describes the porting of a NASA developed Markov reliability analysis tool to a relatively inexpensive IBM-AT. Currently the Markov analysis tool, called SURE, is not widely utilized because it runs in an expensive environment consisting of a VAX, megatex display, and template graphics software. Although substantial savings can be made by running SURE without the expensive graphics, the user friendliness of the tool is dramatically degraded by the lack of graphics. Accordingly a C program using the inexpensive GEM graphics environment has been written. The program is user friendly; it uses menus for option selections and prompts for data entry.

TABLE OF CONTENTS

I. INTRODUCTION

A NASA developed program for computing the death state probabilities for reconfigurable fault tolerant computers forms the basis for this Thesis. Portions of the Semi-Markov Unreliability Range Evaluator (SURE) [Ref. 1] are ported to an IBM-AT environment. The original version of SURE [Ref. 2] used theorems developed in reference 3 which enabled the computation of the death state probabilities of semi-Markov models. The latest version of NASA's SURE uses a generalized method developed by Lee [Ref. 4] and White [Ref. 5] for calculating reliability.

In this Thesis, the essential details of NASA's SURE program will be utilized to develop a relatively inexpensive version of the program, for the IBM-AT environment. The SURE derivative developed in this thesis uses only White's method for analysis of semi-Markov models. White's method uses simple parameters of a model such as means and variance of the transitions to bound the probability of entering a death state of a semi-Markov model. The advantage of using NASA's SURE technique is that the calculated upper and lower bounds of system reliability are algebraic in form, computationally efficient, and normally bound the reliability of the semi-Markov model within 5 percent.

A. RELIABILITY MODELING

In this thesis a graphical method for representing ultrareliable systems will be presented. The current state of the art in electronic component manufacture, produces parts, that used singularly in the production of circuit assemblies do not obtain the reliability standards required for ultrareliable computer systems. Parallel redundancy has been demonstrated to achieve ultrareliable computer systems. Alternatives to massive redundancy, such as hybrid redundancy, use spares and reconfiguration to achieve higher reliability and as a result exhibit both fast and slow transitions. A semi-Markov model representing a hybrid redundancy system composed of three processors and a single spare is shown in figure 1.1.

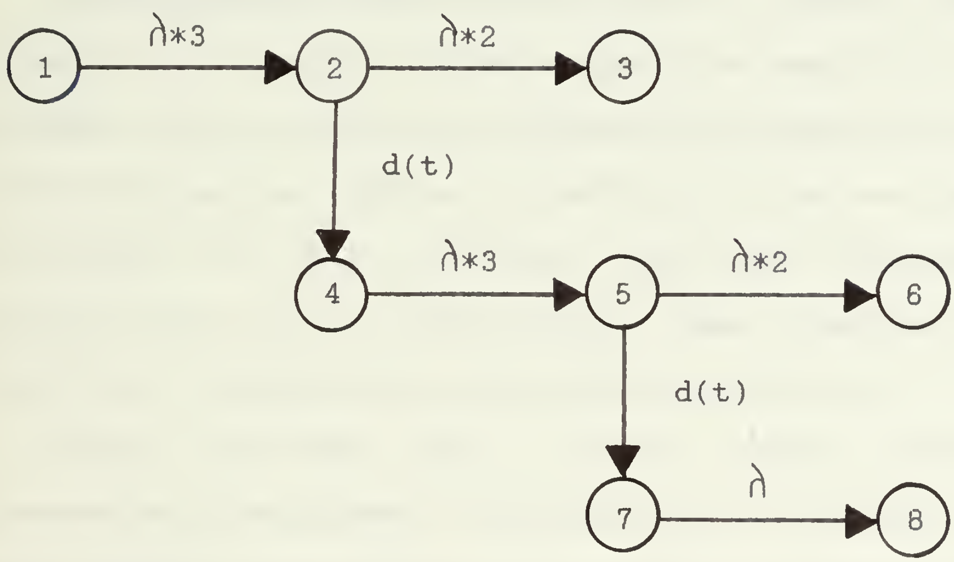

Figure 1.1 Model of a Triad with one Spare
From: reference 1, page 2.

All hybrid models require a combination of slow and fast transitions. A fault or failure is described graphically by a horizontal or slow transition between nodes. Because failure rates are assumed to be constant in a system's operational phase, the fault arrivals are exponentially distributed. The failure rates can be calculated by using the MIL-STD 217D handbook or by experimentation. A constant representing the number of processors currently on line is used in conjunction with the exponential distribution λ to represent a slow or horizontal transition between states. A node represents the current state of the system. The current state is a result of failures and system recoveries leading to the current state. States are represented by numbered circles as shown in figure 1.1.

Hybrid systems have the ability to restore the full voting plane by substituting a spare processor for a failed processor. The recovery is graphically shown as a vertical transition between states. By definition the recovery rate, represented by d(t), is fast. The fast transition is characterized by a conditional mean recovery time, conditional variance and transition probability.

During the time that a hybrid system failure has occurred and the system is attempting to recover, a "race" occurs between recovery and the arrival of another fault. If the system fails to recover, then another horizontal transition will occur. Eventually the hybrid system will

enter what is referred to as a death state as depicted by states 3, 6, and 8 in figure 1.1.

B. THE SURE PROGRAM

The SURE program developed by the NASA Langley Research Center [Ref. 1] is currently running under VMS 3.7 on VAX-11/750 and VAX-11/780 computers. The program was supposedly designed with minimal usage of VMS specific constructs, however this is debatable since many of the constructs are indeed VMS specific and are not easily ported to other machines or operating systems. NASA's SURE is composed of several modules - the computational module which is written in Pascal, the front end module which is written in Pascal and the graphics output module which is written in Fortran, and interfaces to the TEMPLATE graphics library. Because the graphics environment is very expensive and has specific requirements the graphical portion is normally not utilized, by users other than NASA.

The SURE program derivative developed in this Thesis is partially functional on an IBM-AT. The program consists of three modules - the front end module, the computational module and the SURE options module, plus numerous include files. The routines are written in the C language using the Lattice C compiler. Many of the computational routines are a straight Pascal to C conversion. The remaining routines are required to provide the graphical interface for the SURE

derivative program. When completed the SURE derivative program will be able to perform the basic functions of NASA's SURE on a small scale model.

C. PROBLEM OBJECTIVE

Research in fault tolerant computers is being conducted at the Naval Postgraduate School. The need to quantify the reliability of the fault tolerant designs being considered is the driving force behind this thesis. The objective of this project is to implement the SURE program in a relatively inexpensive IBM-AT environment. The program should be user friendly, interactive, and able to perform the same basic operations as the NASA developed SURE program [Ref. 1]. To accomplish the objective, the program was written in the C language and uses GEM (Graphics Environment Manager) [Ref. 6].

Reference 1 is the work of Ricky W. Butler of NASA, Langley. References 1 and 5 are the basis of all the theory discussed in this Thesis. Chapters III and IV present discussions developed on White's technique.

II. PROGRAM DEVELOPMENT

The package developed in this thesis was designed to meet the objectives outlined in the previous chapter. It is an interactive package written in, C, to run on the IBM-AT (Appendix A). The C language was chosen for the following reasons:

1) C programs using the Lattice compiler can interface with the state-of-the-art GEM graphics package.

2) C code is efficient in terms of memory usage and execution speed.

3) C appears to be the micro-computer language of the future, therefore it will be easy for programmers to alter and improve this program in the future.

A. APPROACH TO THE PROBLEM

A combination of top-down and bottom-up programming techniques were used to develop a microcomputer based SURE package. First, the tasks that the program must perform were defined. Then, each task was developed in greater detail until, finally, very specific routines were implemented to perform each task. Of course, a few new tasks were defined and routines created during the development of the program and these new tasks were incorporated with the existing GEM routines.

The tasks which the program performs may be broken down into three basic categories:

1) Window manipulation.

2) Program control.

3) Reliability analysis.

The approach used to accomplish these tasks is discussed in the following subsections.

1. Window Manipulation

The window manipulation task uses GEM Application Environment Services (AES) to build two windows for the display and manipulation of input states and data. GEM AES's subroutine libraries provide routines for a wide variety of tasks, including windowing, monitoring the mouse's movement, displaying system messages and error messages, and drawing objects on the screen [Ref. 6]. The first window (see Figure 2.1) is described as the picture window where all the graphical interpretations of the system are displayed. The second window is referred to as the data window. The picture window is initialized to 60 percent of the screen width; the remaining 40 percent is occupied by the data window.

The manipulation of these windows is controlled by the subroutines that are contained in , the Front-End of SURE (Appendix B). Specific details about the manipulation and editing of these windows can be found in the documentation for GEM [Ref. 6]. Although the author found

the GEM documentation very difficult to use and understand, nevertheless GEM is a very powerful new product that additional documentation should help to make user friendly.

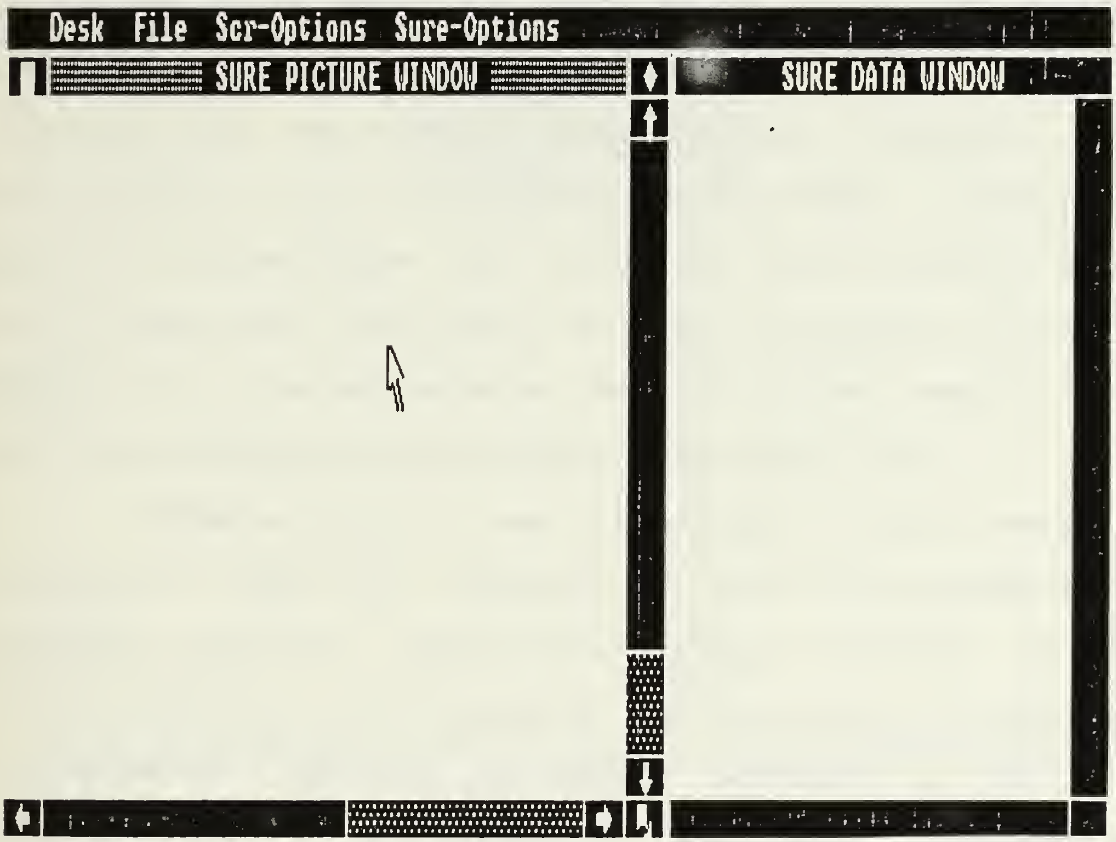

Figure 2.1 SURE Windows

The windows can be manipulated in size, viewing area, and zooming by manipulating "window control areas" located in the border areas of the windows. User interaction with any of the window control areas causes some change to take place, either in the window or the window as a whole.

Additional information about the manipulation of the windows will be discussed in chapter VI.

2. Program Control

The environment that the user works in is modeled after the Apple Macintosh computer. A "mouse" is used to "click" on "windows", pull-down "menus", and otherwise interact with the system. Menus represent groups of SURE options that a user can choose within SURE. To select a menu, the user places the mouse form over the menu's title in the menu bar. This causes the menu to drop down. The menu appears in a rectangle below the menu bar and remains until the user clicks the mouse button (see Figure 2.2).

Program control uses the concept of option menus. A variety of menus are presented to the user throughout the program's execution showing the options available at that time. Sometimes self-management is also used to control the option menus. That is, an option selected by the user sometimes reduces his subsequent options by automatically presenting a predetermined menu.

The menu driven options approach presents options in plain English. As a result the user does not have to know any special language or commands. Self management and plain English options allow the program execution to flow with a minimum number of inputs and far fewer errors.

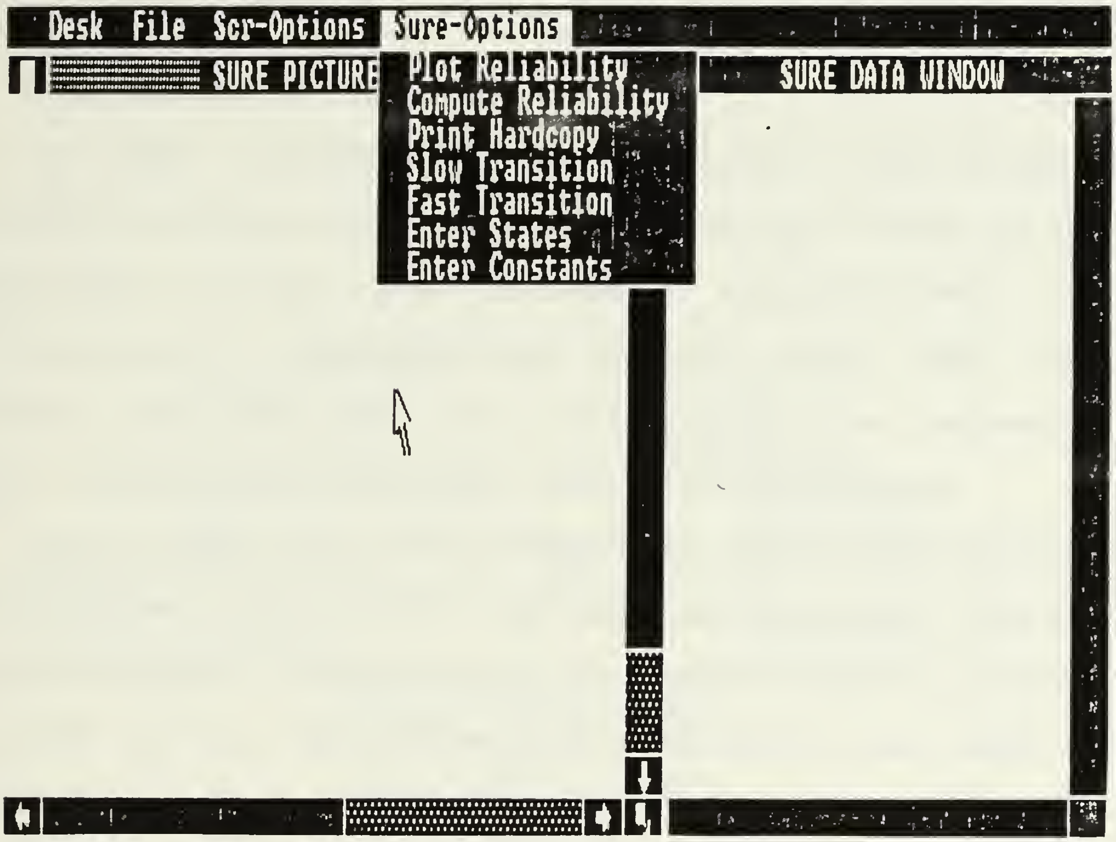

Figure 2.2 Sure Menu

3. Reliability Tasks

The original SURE used both the Lee and White methods to compute the reliability. Although the objective was to transport a graphically enhanced version of the NASA developed SURE tool to an inexpensive environment, the IBM-

AT version uses only White's theorem. Nevertheless the program was designed to allow additional computation methods to be easily added at a later date. The White method is discussed at length in Chapter III.

B. ORGANIZATION OF THE PROGRAM

Low level routines were written based on their output. The input of these routines, then, defined the output of the next higher routine. This process of building upward was continued until ultimately the required input data came directly from the user. For this reason, the parameter input routines form the higher levels of the program.

At all times during the programming process the routines were written using modular programming techniques. In the GEM environment the menu items and dialog boxes are created with the aid of the GEM Resource Construction Set (RCS). A dialog box, which is a special form used in the GEM environment, provides a consistent method of interaction between SURE and the user. After this step is completed the code necessary to use the resource file (menus and dialogs) must be developed. Additional information can be found in reference 6.

The basic organization of the program is shown in Figure 2.3. The shell routines were written first, the input model routines next, followed by the interactive data routines and finally the computation routines.

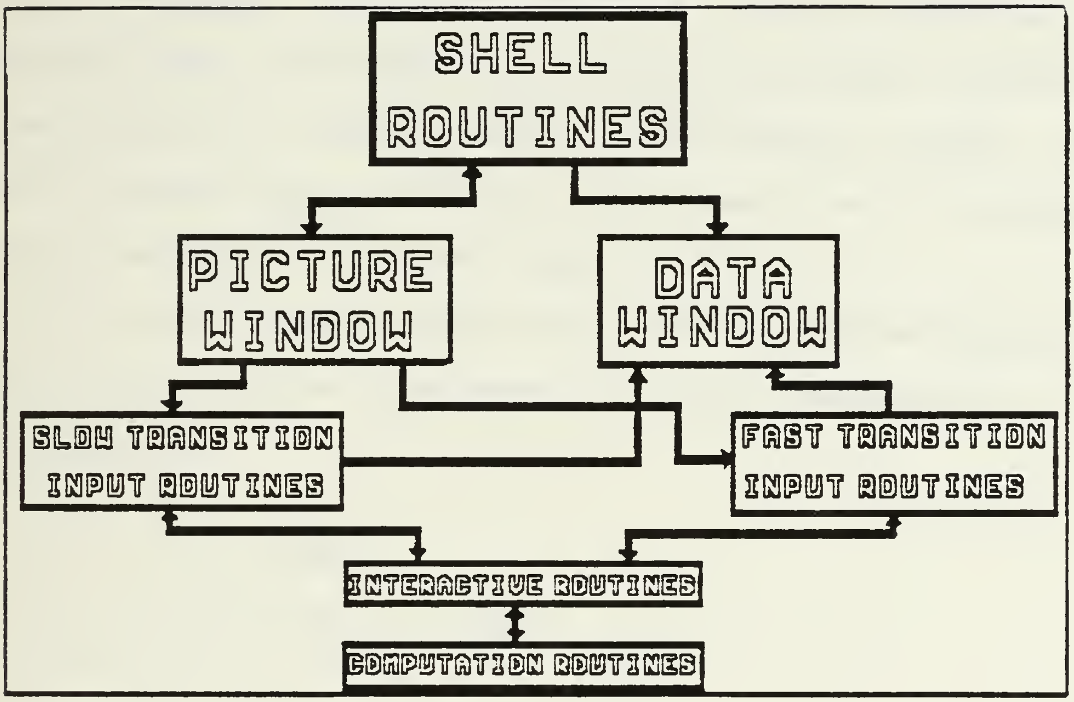

Figure 2.3 Program Organization

III. BOUNDS BASE ON MEANS AND VARIANCES

In this section the Software Implemented Fault Tolerance (SIFT) computer [Ref. 7] figure 3.1 is utilized to describe White's theorem [Ref. 5]. White's theorem provides a graphical means of computing the upper and lower bounds of probabilities of transitioning a given model through all paths to death states. Normally these bounds can be computed within five percent of each other.

In the SIFT model there are a total of sixteen possible paths which must be considered to reach deathstates 4, 8, 11, 14 and 16.

1 -> 2 -> 3 -> 4

1 -> 2 -> 5 -> 6 -> 7 -> 8

1 -> 2 -> 5 -> 6 -> 9 -> 10 -> 11

1 -> 2 -> 5 -> 6 -> 9 -> 10 -> 12 -> 13 -> 14

1 -> 2 -> 5 -> 6 -> 9 -> 10 -> 12 -> 13 -> 15 -> 16

○ - Death State

White's theorem calculates the probabilities of traversing the model through each path. The sum of these probabilities represents the unreliability of the model.

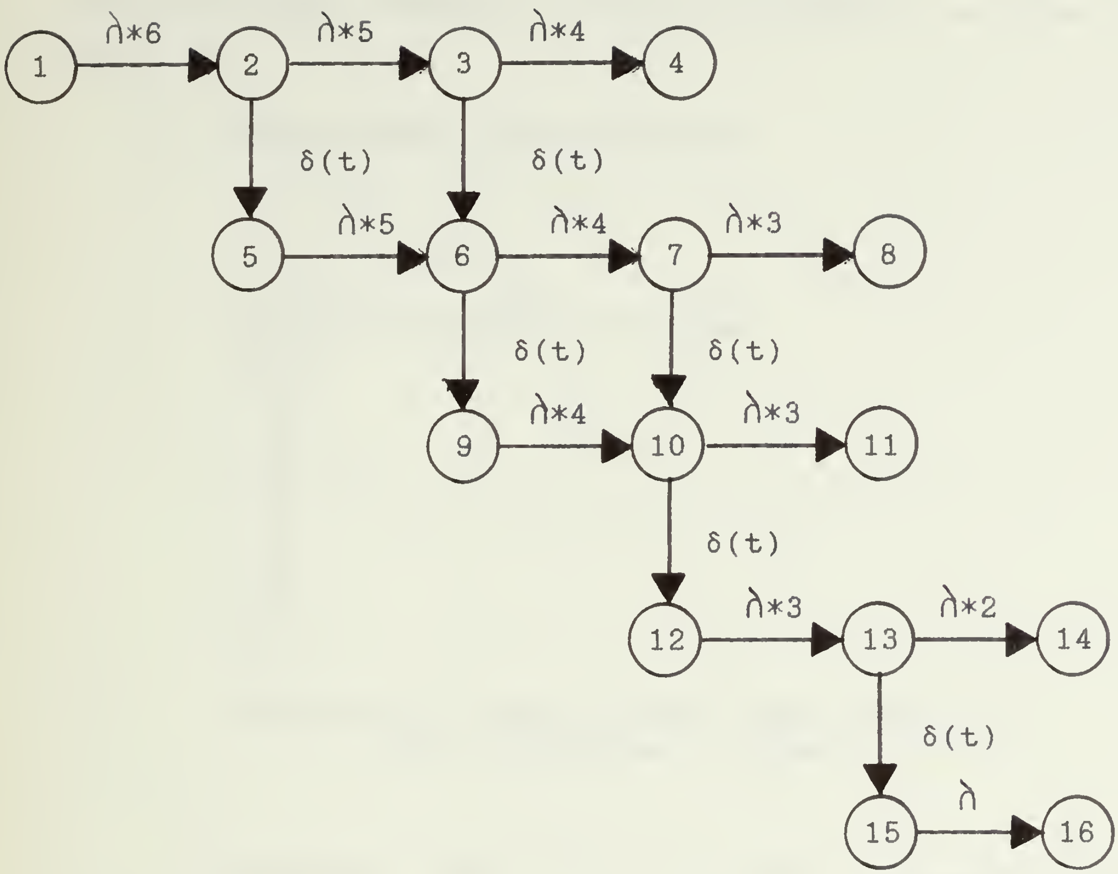

Figure 3.1 Semi-Markov Model of SIFT
From: reference 1, page 4.

A. PATH-STEP CLASSIFICATION

In order to apply White's theorem three classifications of paths through a model must be defined. The classes are defined by the "on-path" and "off-path" transitions from the state. The "on-path" and "off-path" transitions are further defined slow for horizontal transitions and fast for vertical transitions. The term "on-path" will be used to represent the transition currently analyzed. The state and

the transition leaving a state is defined as the "path step".

1. Slow On Path, Slow Off Path

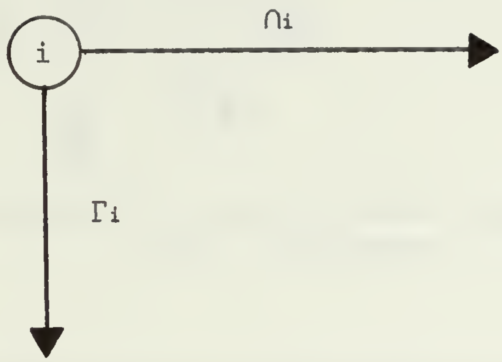

Figure 3.2 Slow on Path, Slow off Path
From: reference 1. page 6.

The first class to be discussed represents slow on-path fault arrival [Fig. 3.2] characterized by the constant failure rate λ_i . This class may contain many off-path slow transitions. The sum of the off-path slow transitions is represented by Γ_i. Transitions 9 -> 10 and 12 -> 13 in figure 3.1 are exponential fault arrivals path steps of this class.

The user should note that in this model there are no fault recovery transitions. The model represents path steps with failure modes only. For electronic circuits with at least one fault recovery reconfiguration, a class three model should be used.

2. Fast On Path, Arbitrary Off Path

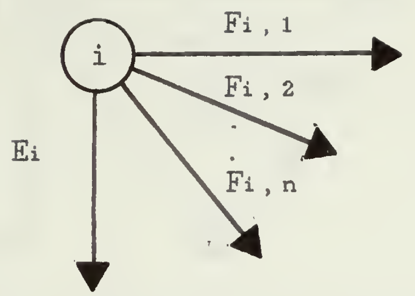

Figure 3.3 Fast On Path, Arbitrary Off Path
From: reference 1, page 6.

This class is characterized by a single on-path fast vertical transition representing fault recovery [Fig. 3.3], E_i represents the sum of all slow transitions. Path steps 6 -> 9 and 7 -> 10 of the SIFT model shown in figure 3.1 are examples of fast transitions. Each fast transition is characterized with a conditional mean, conditional variance and transition probability. If a single recovery transition exists then the transition probability is one, if there is two or more transitions then a fractional transition probability must be assigned. Many off-path fast and slow transitions may exist in this class.

3. Slow On Path, Fast Off Path

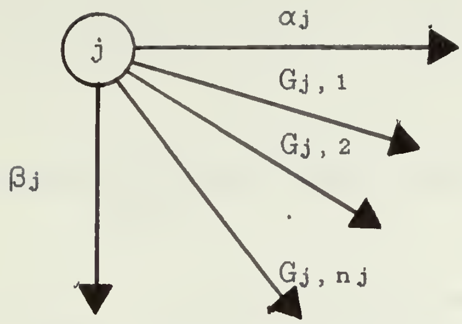

Figure 3.4 Slow On Path, Fast Off Path
From: reference 1, page 11.

This class [Fig. 3.4] covers all transition classes not covered by class one or class two. Specific requirements for this class are that at least one fast transition must be a off-path transition and the slow transition must be on-path. B_j represents the sum of the fault arrival rates for the off-path slow transitions. Paths 7 -> 8 and 10 -> 11 in figure 3.1 represent this class.

B. WHITE'S MULTIPLE RECOVERY THEOREM

White's theorem as presented in reference 1, bounds the upper and lower value of the probability D(t) (see Eq. 3.1) of entering a death state during mission time T, subsequent to k class one path steps, m class two path steps and n class three path steps :

$$LB \leq D(T) \leq UB \tag{3.1}$$

where

$$UB = E(T) \prod_{i=1}^{m} p(F_i) \prod_{j=1}^{n} \alpha_j \; u(H_j) \tag{3.2}$$

$$LB = E(T-\delta) \prod_{i=1}^{m} p(F_i)\left[1-\epsilon_i \; u(F_i) - \frac{u^2(F_i) + \sigma^2(F_i)}{r_i{}^2}\right] \prod_{j=1}^{n}$$

$$\alpha_j \left\{ u(H_j) - \frac{(\alpha_j+\beta_j)[u^2(H_j)+\sigma^2(H_j)]}{2} - \frac{u^2(H_j)+\sigma^2(H_j)}{s_j} \right\} \tag{3.3}$$

$$\delta = r_1 + \ldots + r_m + s_1 + \ldots + s_n$$

D(t) = Probability of Entering a Death State

UB = Upper Bound

LB = Lower Bound

U = Conditional Mean

σ^2 = Conditional Variance

P(Fi)= Transition Probability

E(T) = the probability of traversing a path consisting of only the class 1 path steps within time T.

IV. TRANSIENT AND INTERMITTENT MODELS

White's theorem as discussed in chapter III is designed to calculate the probability of entering a pure death state of a semi-Markov model. Unfortunately the analysis of an electronic circuit may not end at a set of finite paths through the model. Occasionally hybrid models under analysis will require the modelling of paths which repeat as a result of intermittent or transient faults.

Accordingly, several models that may be used to represent this class of fault will now be discussed.

The first model of a transient fault can be represented as shown in figure 4.1.

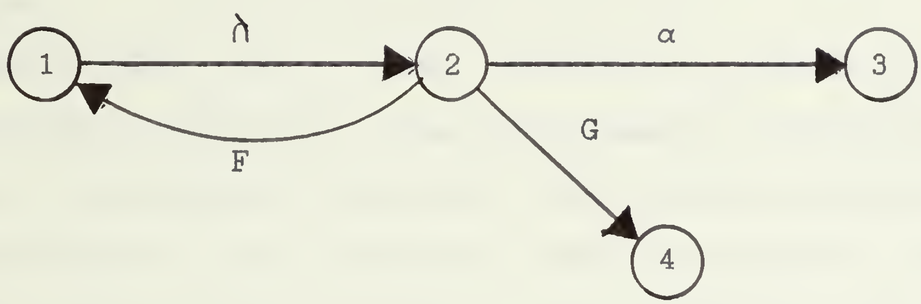

Figure 4.1 Transient Fault Model
From: reference 1, page 13.

The constant fault arrival rate λ represents faults of a transient nature in the model. The distribution function F(t) represents the time that the transient fault may exist in the hybrid circuit being modelled. The hybrid circuits being modelled will try to recover by circuit

reconfiguration. The reconfiguration process has a distribution represented by G(t). An infinite series of paths results from using this model [Fig. 4.2].

```
1 -> 2 -> 3
1 -> 2 -> 1 -> 2 -> 3
1 -> 2 -> 1 -> 2 -> 1 -> 2 -> 2 -> 3
1 -> 2 -> 1 -> 2 -> 1 -> 2 -> 1 -> 2 -> 3
   .
   .
   .
```

Figure 4.2 Infinite Paths
From: reference 1, page 13.

As the length of the path increases the significance of the probability of entering the pure death state 3 in the model decreases. By limiting the number of times a particular path or loop is repeated computational efficiency can be achieved.

NASA's SURE has a user controlled special constant called TRUNC which will limit the number of times that a path is traversed in the model. This special constant will be included in the completed SURE derivative program. Another interrelated constant is the PRUNE constant. It is user-specifiable and will terminate the computation of all paths which fall below the probability specified by the PRUNE constant. The PRUNE constant effects all paths in the model inclusive of paths that do not repeat. NASA recommends that users experiment with different values for TRUNC and PRUNE to ensure convergence.

The next model to be discussed is the model representing intermittent faults [Fig. 4.3]. The fact that an intermittent fault does not disappear differentiates it from the transient fault. A fault that is not active is represented by Q in the intermittent model.

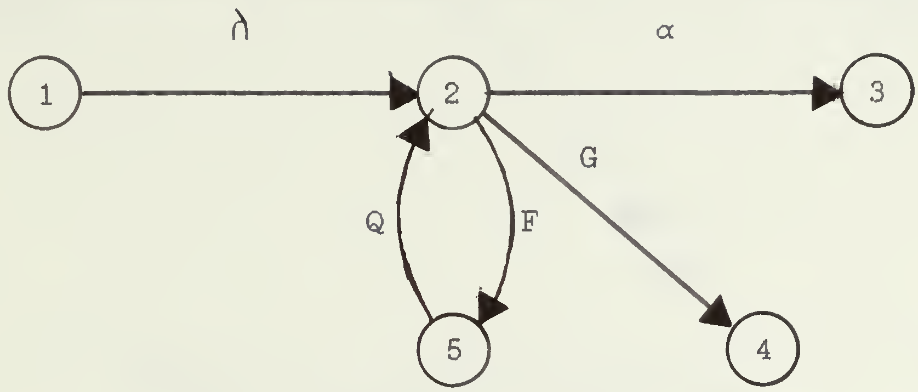

Figure 4.3 Intermittent Faults
From: reference 1, page 15.

This model shown in figure 4.3 appears very similar to the previous model, the model is however computationally very different. In the intermittent model a loop is formed on a vertical or fast transition, which may cause a very slow convergence.

The recommended approach [Ref. 1] for circuits exhibiting slow convergence is to model the circuit with the collapsed transition model shown in [Fig. 4.4]. In the collapsed transition model G* represents the conditional mean, conditional variance of the recovery time distribution

and transition probability of a path step attempting to recover from an intermittent fault. These values can be found experimentally, or calculated from F, G and Q of the previous model.

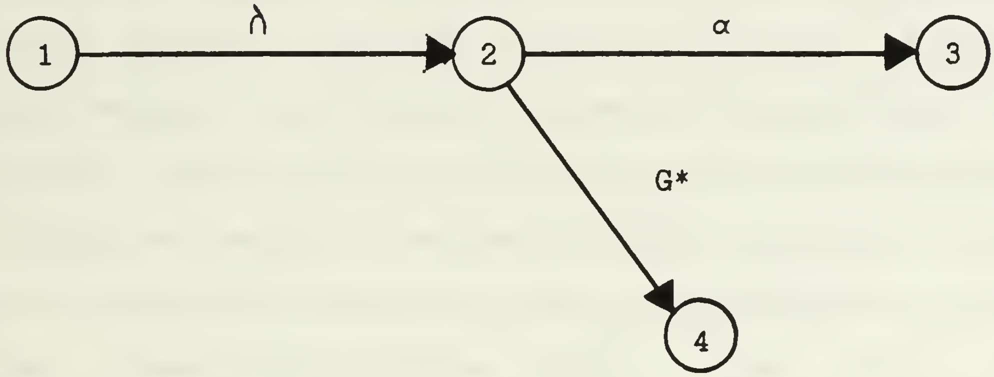

Figure 4.4 Collapsed Transition
From: reference 1, page 16.

The analysis given in this section came from reference 1 it can be applied to many similar models to demonstrate that convergence will occur. Users desiring more information are directed to reference 1.

V. THE SURE USER INTERFACE

The SURE derivative program creates a graphical environment in which a semi-Markov model can be graphically input with a mouse and high resolution display. A simple graphical method is utilized to enumerate the fast and slow transitions. This is accomplished using many of the routines and graphical symbols resident in GEM. Graphical symbols such as circles and arrows are used to enumerate the states and the transitions between states. Menu options are selected by the user initiating very specific dialog boxes to appear in the center of the screen. The dialog boxes require the user to interact, and form the basis for direct data input to the SURE program. The input data which is input by way of the dialog boxes is stored in data structures for use by the SURE program. The graphical language required by the user for interaction with the SURE program will be discussed in detail in this section.

A. BASIC PROGRAM CONCEPT

As the first step in describing a semi-Markov model the SURE user must assign state locations in the picture window. Assignment of state numbers is done automatically as the state locations are defined. State assignment numbers begin at one and are incremented to a maximum of 100. The semi-

Markov model description is continued by enumerating all the transitions. As described in the previous sections, each transition is described as being either slow or fast. Consequently, there are two different menu selections used to enter transitions - one for slow transitions and the other for fast. If the transition is slow, then the slow menu selection is used. The user simply moves the mouse into the Sure-Options part of the Main Menu. A drop down menu will appear. The user then positions the mouse on the appropriate entry and clicks the mouse (see Figure 5.1).

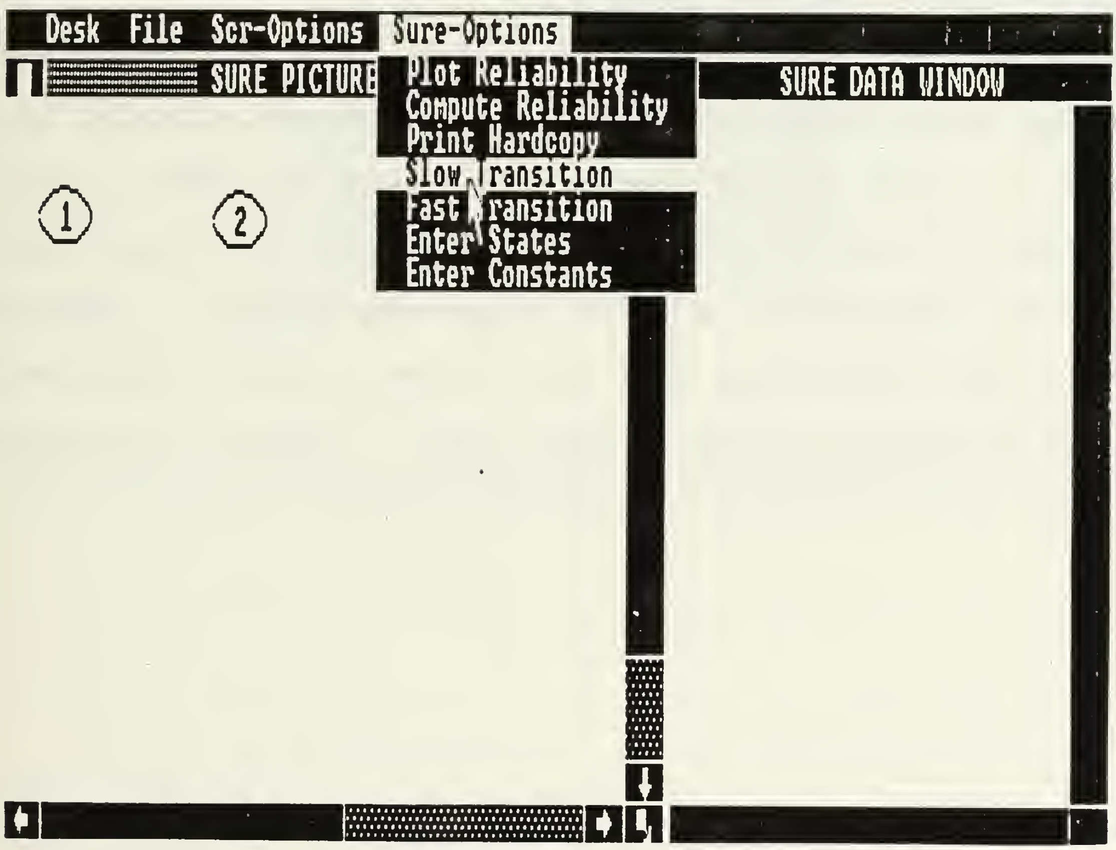

Figure 5.1 Slow Transition

A slow transition is the most trivial transition to draw graphically. The routines required to draw and label the graph are located in Appendix C. The routines necessary are self documenting and should be easy to follow.

This menu selection causes a dialog box to appear in the center of the screen (see Figure 5.2). A slow transition from 1 to 2 can then be defined by typing in the exponential transition rate. This value can be defined using previously defined constants or by typing in the numeric values.

Figure 5.2 defines a slow exponential transition from state 1 to state 2 with rate 0.0003. If the transition is fast, then the fast transition menu selection is made. This selection causes a fast transition dialog box to appear in the center of the screen (see Figure 5.3). A fast transition from state 2 to state 4 is defined by entering the mean, standard deviation and the probability of the transition. In both the slow and fast transitions the values entered will appear on the screen as shown in Figure 5.4.

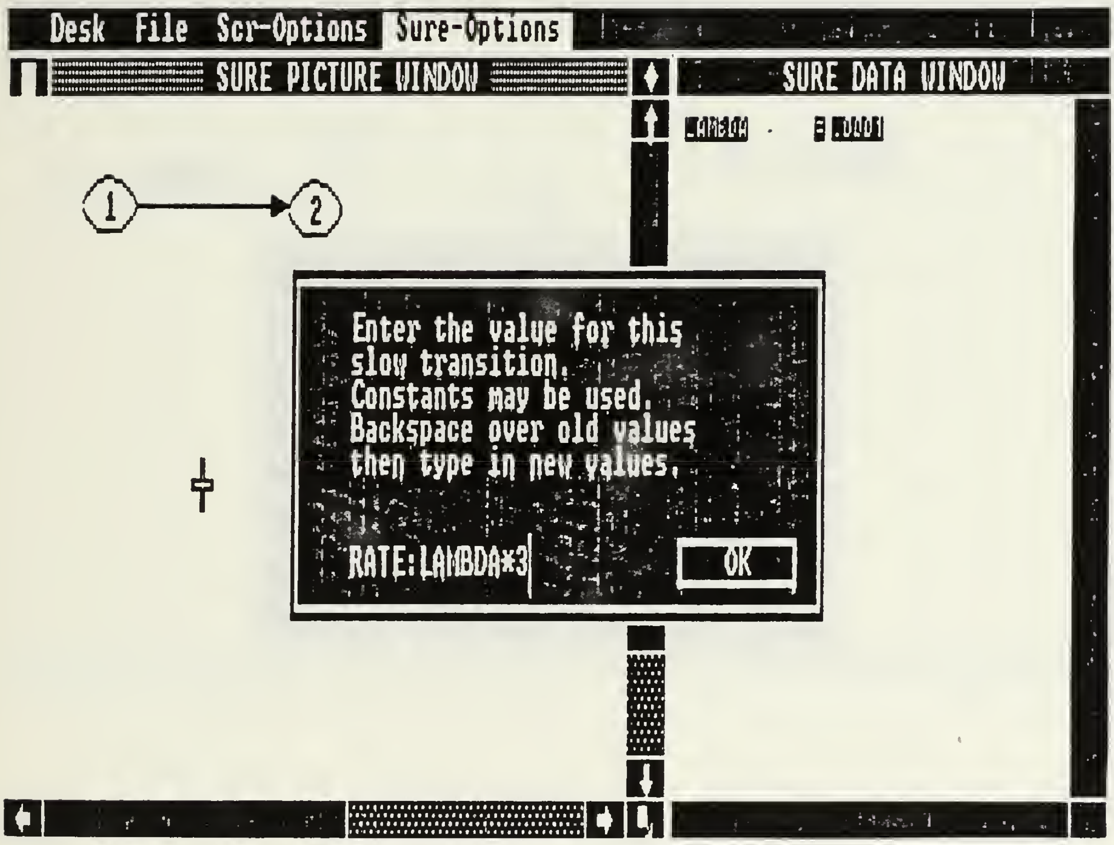

Figure 5.2 Slow Transition Dialog Box

Figure 5.3 Fast Transition Dialog Box

Figure 5.4 Example Slow, Fast Transition

B. SURE MODEL DEFINITION SYNTAX

The transition-description menu selection described above are the only essential ingredients in the SURE graphical language. However, the flexibility of the SURE program has been increased by adding several features commonly used by menu driven programs.

The SURE user may equate numbers to identifiers. Thereafter, these constants identifiers may be used instead of the numbers. For example,

LAMBDA = 0.0052;

can be entered by selecting the menu item labeled enter constants, again a dialog box will appear in the center of the screen. The user may then enter the identifier and it's value (see Figure 5.5).

The user should use a little forethought before initiating and defining the constants. When the dialog box appears the name of the constant is normally selected and should be entered first followed by its value, however either can be defined first. The editable fields within the dialog box are selected by moving the mouse to the desired field and clicking once. The desired values can then be entered by way of the keyboard. The user may need to backspace over the last items that were entered.

Constants may also be defined in term of previously defined constants:

GAMMA = LAMBDA*10;

In general the constant is identified by a string of up to eight letters, digits, and underscores (_) beginning with a letter, with a value defined as an arbitrary mathematical expression.

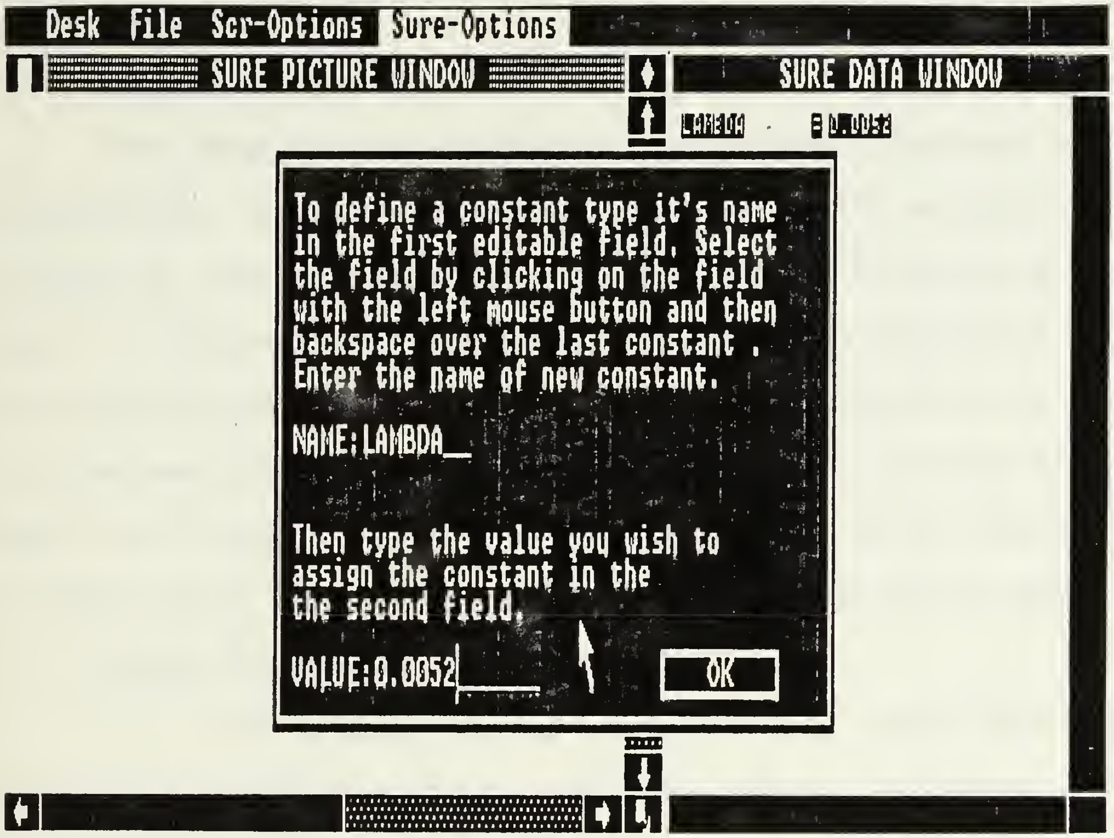

Figure 5.5 Enter Constants Dialog Box

When specifying transition or holding time parameters in a dialog box, arbitrary functions of the constants may be used. The following operators may be used:

+ addition
- subtraction
* multiplication
/ division

Expressions may be grouped by using ().

The following are permissible expressions:

```
.00001
alpha*1.2
alpha*1.2 + alpha*12
```

1. <u>Entering States</u>

The states are entered by selecting the menu item labeled enter states. The user then carefully selects the position of the states in the picture window by placing the mouse in the desired location. The user then depresses the left mouse button, a state is then drawn and labelled in the location specified. The program will not allow a state to be drawn too close to another state nor will it allow the user to draw a state too close to the boundaries of the window.

2. <u>Slow-Transition Description</u>

A slow transition is completely specified by selecting a slow transition from the menu, clicking the mouse button on the source state, and then clicking the mouse button on the destination state. The resulting transition is displayed in the picture window as an arrow connecting the source state to the destination state. The dialog box will appear and the user may enter the desired rate for this slow transition. The following will appear in the data window:

```
"source"  TO "dest" = "rate"
```

where "source" is the source state, "dest" is the destination state, and "rate" is any valid expression defining the exponential rate of the transition. In the notation of the theory section we have

$i,j = \lambda_i$;

3. Fast-transition Description

To enter a fast transition the appropriate menu item is selected, then the user designates the source and destination as described above. A dialog box appears and again the appropriate information is entered. The following syntax will appear in the data window.

"source" TO "dest" = "mean" ,"stddev", "fract"

where

"mean" = an expression defining the conditional mean transition time, u(F).

"stddev" = an expression defining the conditional standard deviation of the transition time, $\sigma(F)$.

"fract" = an expression that defines the transition probability, p(F)

and "source" and "dest" define the source and destination states, respectively. In the notation of the theory section, we have

$i,j = < u(F_i), \sigma(F_i), p(F_i) >$;

The third parameter "fract" is required to define the transition probability. If there is only one recovery

transition the probability is 1.0. If there exists more than one recovery transition the sum of the probabilities from each state should total 1.0, the user must ensure this.

C. SURE COMMAND SYNTAX

The SURE menu is broken down into four sub-menus, Desk, File, Scr-Options and Sure-Options (see Figure 5.6).

DESK	FILE	SCR-OPTIONS	SURE-OPTIONS
About Sure	Load	Pen/Eraser	Plot Reliab
Calculator	Save	Erase Pic	Compute Reli
Clock	Save AS	View	Print Hard
	Abandon		Slow Trans
	Quit		Fast Trans
			Enter Stat
			Enter Cons

Figure 5.6 Sure Menus

Each entry will be discussed in the following sections.

1. Desk

The sub-menu desk contains menu items that are not required by SURE. The menu items are listed below:

About Sure - Produces a dialog box listing information about the author of SURE.

Calculator - For convenience, a calculator will appear in the center of the screen allowing the user to obtain the value of an arbitrary expression.

Clock - System time will appear on screen

2. File

The sub-menu File contains the menu items for the loading and storing of SURE specific information. Each menu item is discussed below:

Load	-	The load selection is used to load a SURE run that was previously saved.
Save	-	Saves the file for current SURE run. The user must have previously named the file by means of a load or Save As.
Save As	-	Saves the present SURE run under a new filename.
Abandon	-	Reverts to the last-saved version of the users file. If the user is dissatisfied with changes made , Abandon clears the changes and gives the user a fresh copy.
Quit	-	Takes the user back to the GEM Desktop.

3. Screen-Options

This sub-menu controls the operations performed in the picture window and data windows. The screen options are discussed below:

Pen/Eraser	-	The selection allows the user to select the small, medium and large states and colors as desired.
Erase Pic	-	Erases the currently displayed information from the picture and data windows.
View	-	Allows the picture window to be displayed in several expanded and non-expanded modes.(To be added Later)

4. Sure-Options

This sub-menu contains the main SURE user interface menu items. It should be used continuously during a SURE session. Menu items are discussed below:

Plot Reliab	-After a Compute Reliability, this item can be selected to plot the output in the picture window.(To be added later)
Compute Reli	-After a semi-Markov model has been fully described to the SURE program, this item is used to initiate the computation.
Print Hard	-This item is used to print the screen to a printer.(To be added later)
Slow Trans	-Used to define a slow transition. Causes a interactive dialog box to appear in the center of the screen. The user types in the value for the slow transition.
Fast Trans	-Used to define a fast transition. Causes a interactive dialog box to appear in the center of the screen. The user types in the values for the fast transition.
Enter Stat	-Used to enter the states. Causes the SURE program to prepare to receive the state locations specified by the user. The user defines state locations by positioning the mouse cross-hairs and pressing the button.
Enter Cons	-Used to enter the user defined constants. Causes an interactive dialog box to appear in the center of the screen. The user may define new constants or any of the Special constants discussed below.

5. <u>SURE Derivative Special Constants</u>

NASA's SURE has many special constants (see TABLE 5.1) which interact with the user to provide special program control. The definitions and the names of the constants used by NASA (Ref. 1, page 25 & 26) are given in TABLE 5.1. Several of these special constants such as TRUNC and PRUNE were previously discussed. The completed SURE derivative program will implement several of these constants as follows:

TABLE 5.1 SPECIAL CONSTANTS

TIME	=	25;	Sets the mission time to 25. The default TIME is 10.
POINTS	=	25;	Indicates that 25 points should be calculated/plotted over the range of the variable. Default value is 25.
PRUNE	=	.1;	Sets pruning level to .1. The program will stop searching a path after its current probability becomes less than the specified level. The default is 0.0.
WARNDIG	=	2;	Sets the number of digits accuracy desired in the upper bound to 2 when the PRUNE command is in use. The default is 1.
TRUNC	=	2;	Sets truncation point at 2. Default is 3. If an infinite loop is found in the graph this determines the maximum number of times that the loop will be traversed.
LBFACT	=	20;	Sets the k_i and k'_j constants in White's theorem to 20.
LIST	=	3;	Sets the level of information to be sent to the Data window. Default is 2.

Adapted from: reference 1, page 25 & 26.

VI. EXAMPLE SURE SESSIONS

A. OUTLINE OF A TYPICAL SESSION

The SURE program was designed for interactive use. It uses a graphical approach to describe a semi-Markov model. The following method of use is recommended:

1. Select the desired state size, using the pen/erase menu item. The default size is medium. All graphical sizing aids are presently designed to operate in the medium mode.

2. Enter all the Special constants.

3. Enter all the user defined constants.

4. Use the mouse to carefully enter the state locations. Verify that the model appears in the desired form. If the model is not in the desired form erase the model by using erase picture.

5. Carefully define the transitions by selecting either a slow or fast transition. Define the source state by clicking once on the state, define the (dest) destination state by clicking once on the state. The appropriate state is selected if the mouse location is within 18 pixels of the center of a state. Enter the values to describe the transition.

6. Select the compute reliability menu item.

B. EXAMPLES

The following examples illustrate direct interactive SURE sessions. The actual graph is given first then the SURE program analysis. The final results are simulations since the computation module which calculates the upper and lower bounds is still being debugged.

1. <u>Example 1</u>

This session illustrates direct interactive input of a simple semi-Markov model (see Figure 6.1).

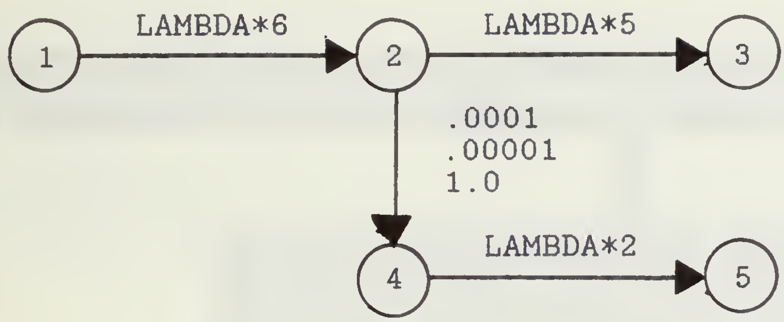

Figure 6.1 Simple Model

The following sequence of Figures is exactly what the SURE user would see during the execution of this example. Figure 6.2 is the initial screen displayed when the the Sure program is selected.

The sequence to follow is not critical to the operation of the SURE program. It is simply intended as a guideline. The sequence can be rearranged in practically any order. The user may add states at any time during the initial diagram description. If during the process of describing the graphical model a constant is changed, that defines a transition, the model will not display the change but the program will use the latest value of the constant in it's computations.

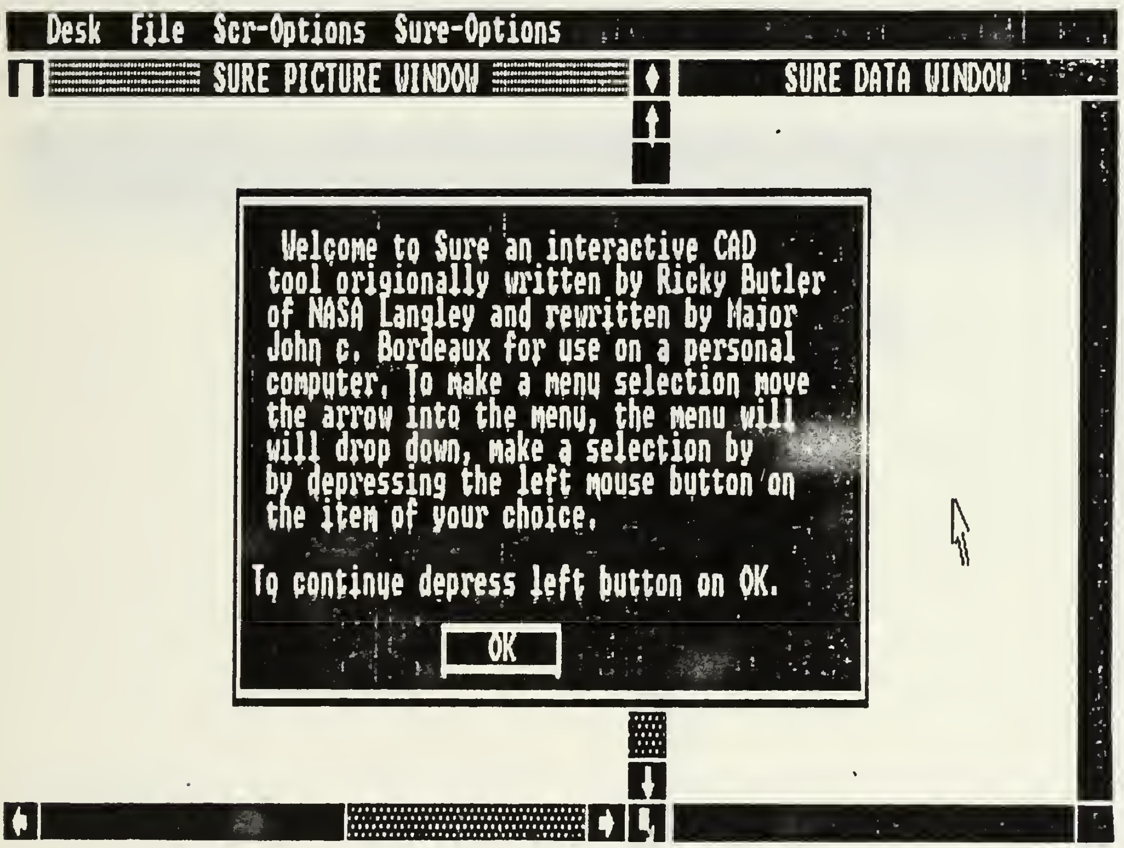

Figure 6.2 SURE Initial Display

Next, the constants for this example are entered by selecting enter constants. Figure 6.3 illustrates the display after all the constants have been entered.

Figure 6.3 Example 1 Constants

The states are the next items to be added to this SURE problem. As discussed earlier care must be taken in selecting the locations for the states. The graph shown in

Figure 6.4 was produced while trying to keep in mind the graph depicted in Figure 6.1.

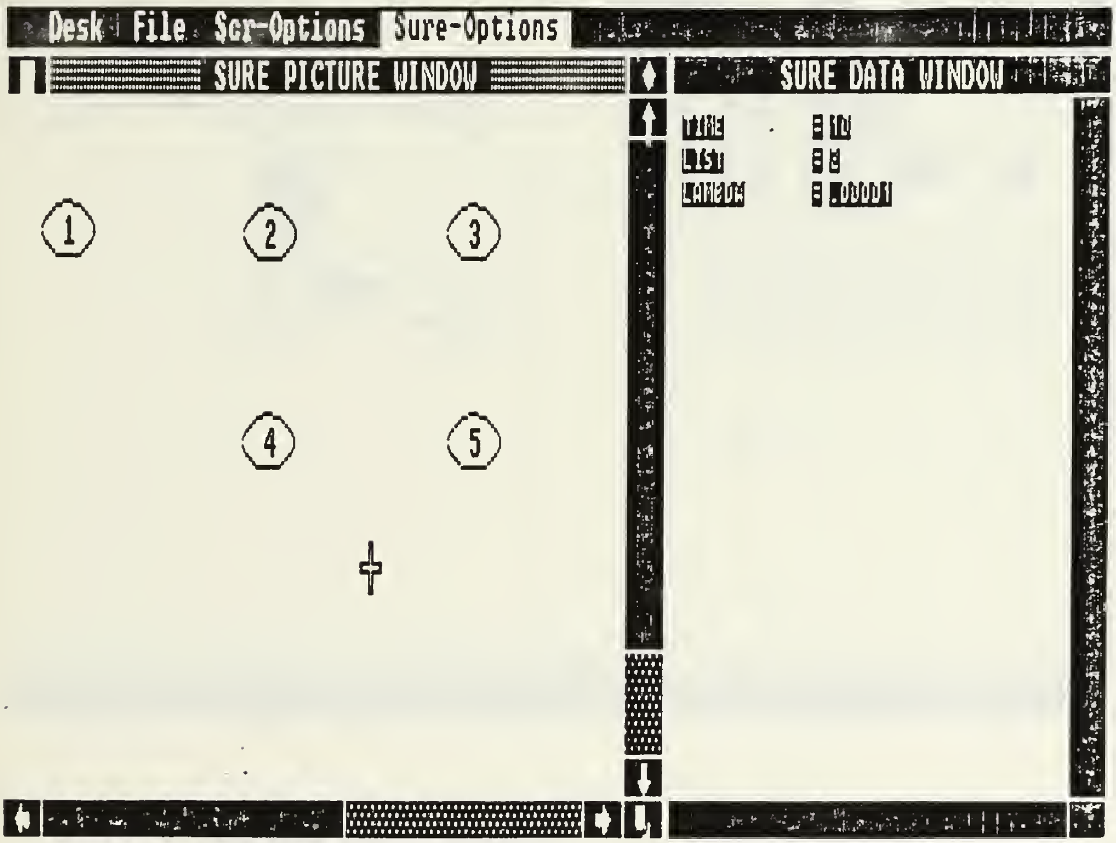

Figure 6.4 Example 1 States

After the states have been entered, the transitions are entered as discussed in Chapter five (5). The completed graph is shown in Figure 6.5.

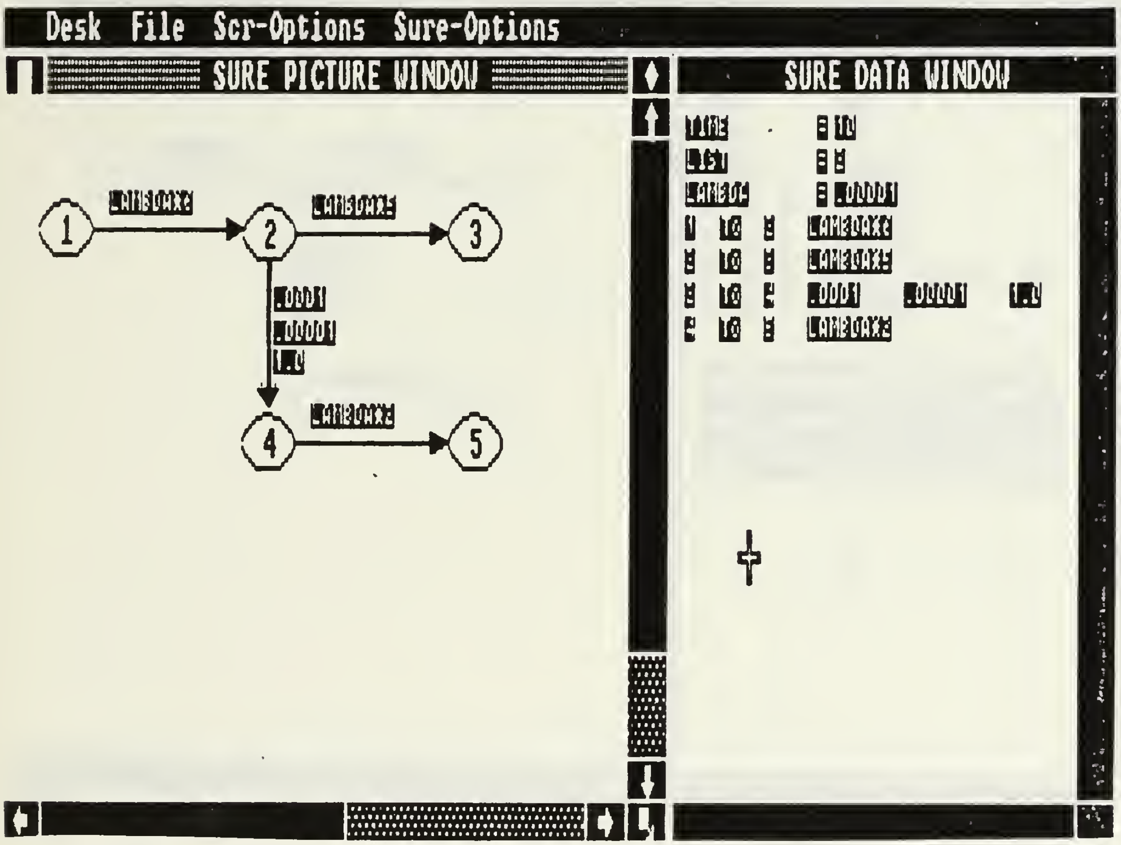

Figure 6.5 Example 1 Complete Graph

The next step is to do the calculations. This is accomplished by making the menu selection labeled Compute Reliability. The complete example 1 results are displayed in Figure 6.6.

Figure 6.6 Complete Example 1
(Calculations have been Simulated)

2. Example 2

The following example illustrates the use of SURE to solve a model of a triplex system with one spare (see Figure 6.7). The plot reliability function is also displayed.

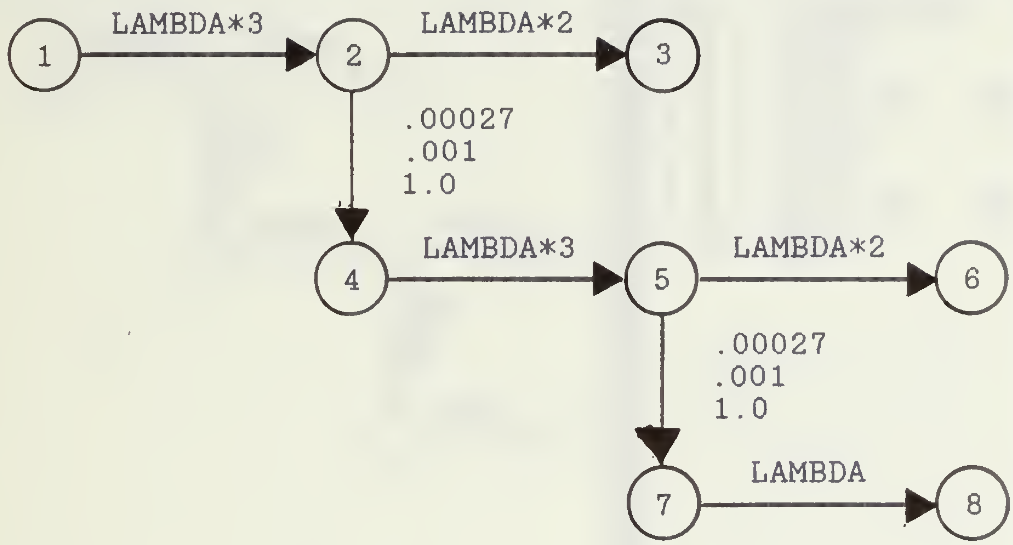

Figure 6.7 Example 2 Model

Figure 6.8 shows the model after it has been entered into the SURE program. Again note that an effort was made to keep good alignment and to represent the model as shown in Figure 6.7.

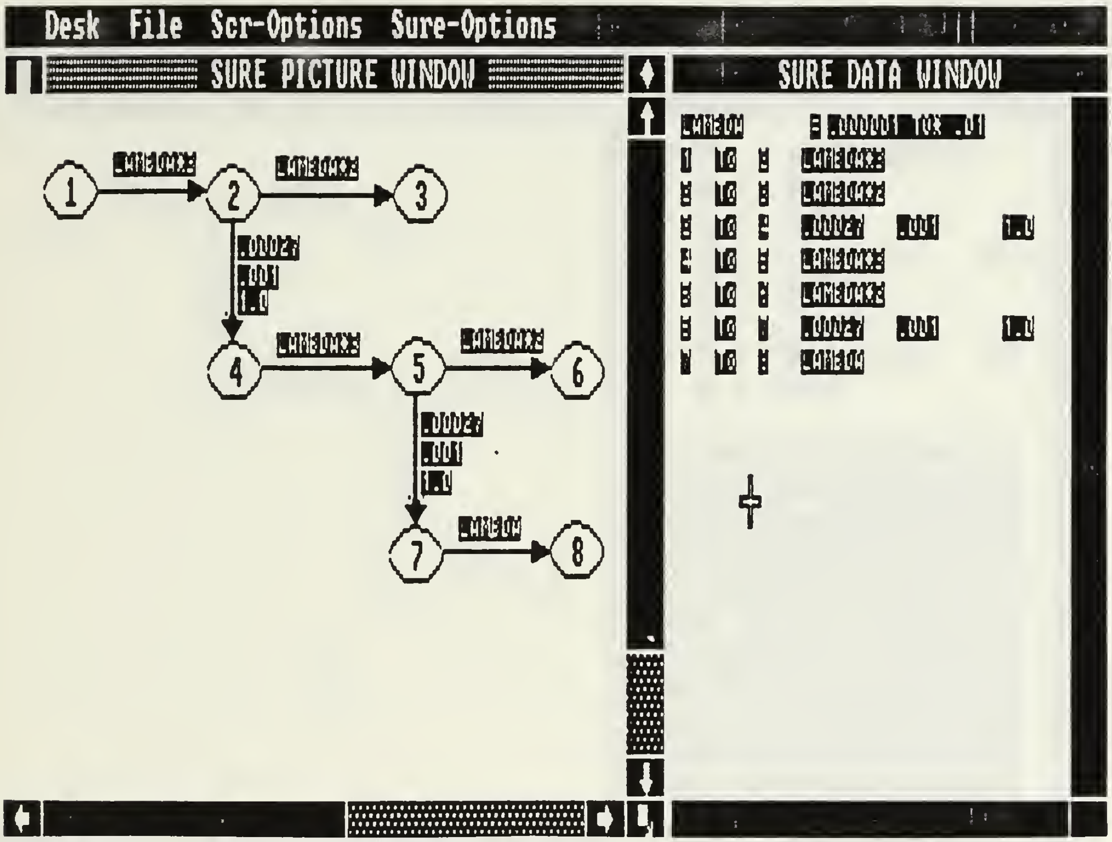

Figure 6.8 SURE Example 2 Entered

Figure 6.9 shows the completed example 2 problem. The problem can be saved for future use by using the functions described earlier under the sub-menu titled File.

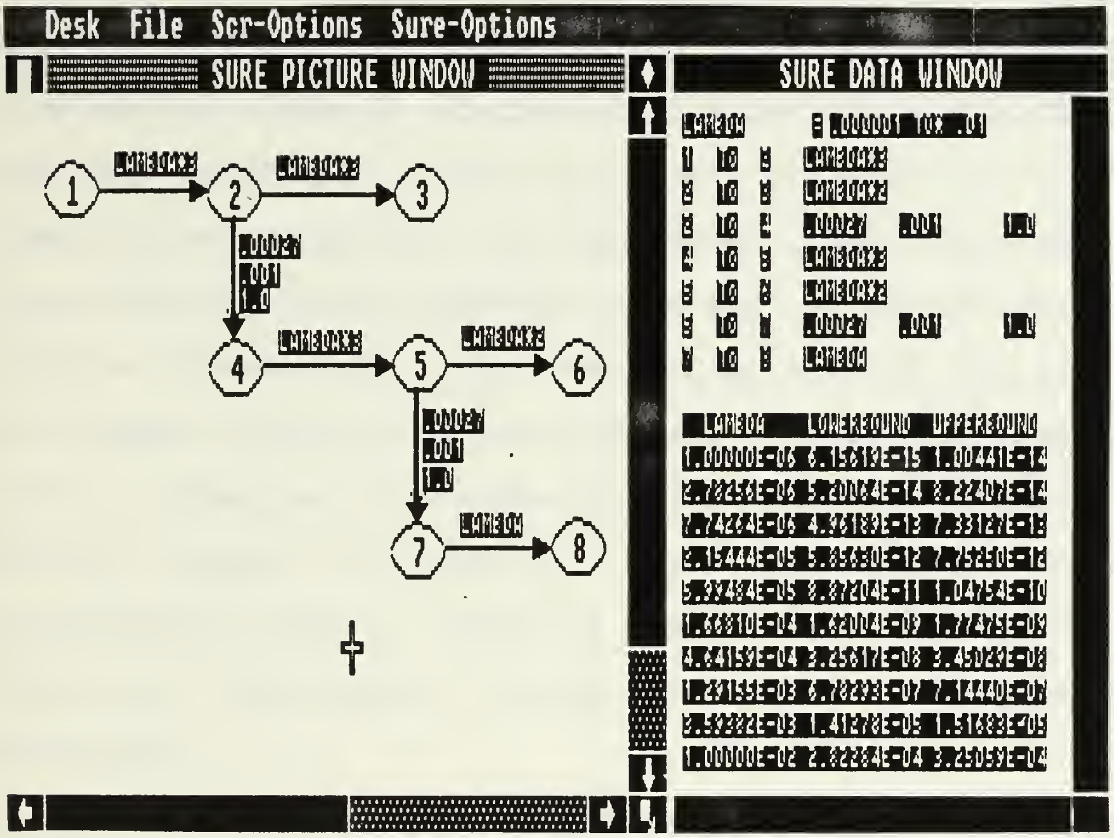

Figure 6.9 Completed Example 2
(Calculations have been simulated)

VII. CONCLUSIONS

A. SUMMARY OF RESULTS

The SURE program is a flexible, user-friendly, interactive graphical tool, modeled after the SURE program developed by Ricky W. Butler of Langley Research Center Hampton, Virginia. The program provides a rapid computational capability for semi-Markov models useful in describing the fault-handling behavior of fault-tolerant computer systems. The only modeling restriction imposed by the program is that general recovery transitions must be fast in comparison to the mission time. The SURE reliability analysis method utilizes a fast approximation theory developed by Allan L. White of PRC Kentron, Inc.. These upper and lower bounds are typically within 5 percent of each other.

Although the approximation theory does not explicitly deal with semi-Markov models that are not death processes, the SURE program utilizes simple path truncation strategies to enable the analysis of such models.

B. RECOMMENDATIONS

It is recommended that additional work be done on the program to remove the following bugs which presently exist in the program.

1) State locations are determined by storing the pixel coordinates in a structure called STATE_LOC. These coordinates are determined by the location of the mouse in the picture window when a state location is specified by the user. The program will currently operate using pixel locations of the picture window when the picture window has not been scrolled.

2) The parser required to implement the use of all the special constants has not been included in the listings. The final listings will be available on disk.

3) The GEM package currently uses a very crude method of writing to the screen. The GEM subroutine v_gtext(), requires all text to be in a character string. Additional work should be done to create a better method of displaying floats and integers.

APPENDIX A

GETTING STARTED

HARDWARE REQUIREMENTS

The recommended hardware for the SURE program is an IBM PC/XT or AT (or 100% compatible) with 512K RAM, a graphics card, and a mouse.

Supported graphics card include:

- Monochrome Hercules Graphics Card(TM)
- IBM Color Graphics Card (in monochrome mode)
- IBM Enhanced Graphics Card

Supported mice are PC Mouse by Mouse Systems(TM), the Microsoft Mouse, Summary graphics Tablet, and keyboard.

SOFTWARE REQUIREMENTS

In order to use SURE you must have the GEM Programmer's Toolkit(TM) installed on your machine. To install the software on your IBM PC/XT or AT, follow the instructions included with GEM, or do the following:

1. Load DOS into your computer.
2. Place the GEM SYSTEM MASTER in drive A, type GEMPREP, and press ENTER. Follow the instructions carefully, answering any questions you are asked.
3. Create a subdirectory called TOOLS and copy the disks with GEM PROGRAMMER's TOOLKIT to that area.
4. Copy the SURE program disk into that area.

5. The items listed below are required for the user who wishes to edit the SURE program.

   - PC DOS version 2.0 or higher

   - the lattice C compiler (the user may use another compiler but extensive rewriting of the GEM bindings may be required)

   - an assembler, linker, and librarian such as those provided in DR Assembler Plus Tools (RASM-86, LINK-86, LIB-86)

USING SURE

The SURE application can be executed by/from the GEM Desktop, by doing either of the following:

- Double-click on the SURE icon.

- Select the SURE icon and then choose the Open command from the File Menu.

Note: An icon is a GEM symbol.

APPENDIX B

FRONT END OF SURE LISTING

The SURE routines that contain the code required to manage the windows and menus are contained in this Appendix. The code was developed by using a example of an application that came with GEM called Demo. Considerable modification of the original code has taken place. Most of the code conversion was done to add the second window called the data window. The window called the picture window was referred to as the work_area in the demo program.

Demo contains several routines which are not currently being used by SURE. These routines were retained for future development of the SURE program.

```
/**********************************************************************/
/*              File: SURE.C                                          */
/**********************************************************************/
/*                                                                    */
/*              SSSSSS     UU   UU    RRRRR     EEEEEE                */
/*              SS         UU   UU    RR  RR    EE                    */
/*              SSSSSS     UU   UU    RRRRR     EEEEEE                */
/*                  SS     UU   UU    RR  RR    EE                    */
/*              SSSSSS      UUUUU     RR  RR    EEEEEE                */
/*                                                                    */
/**********************************************************************/


/**********************************************************************/
/*   Author:   MAJOR JOHN C. BORDEAUX  U.S.M.C.                       */
/*   PRODUCT:  MARKOV ANALYSIS TOOL                                   */
/*   Module:   REVISION OF DEMO,  Version 1.1,FROM DRI                */
/*   Version:  APRIL 1986                                             */
/*                                                                    */
/**********************************************************************/


/*------------------------------*/
/*   includes                   */
/*------------------------------*/

#include "portab.h"                    /* portable coding conv      */
#include "machine.h"                   /* machine depndnt conv        */
#include "obdefs.h"                    /* object definitions          */
#include "treeaddr.h"                  /* tree address macros         */
#include "gembind.h"                   /* gem binding structs         */
#include "sure.h"                      /* SURE  apl  resource         */
#include "math.h"
#include "limits.h"
#include "optstrct.h"
/**/                                   /*   file offsets              */


/*------------------------------*/
/*  defines                     */
/*------------------------------*/

#define    MAXSTATE     100
#define    MAXTRANS     200
#define    ARROW        0
#define    HOUR_GLASS   2

#define    DESK         0

#define    END_UPDATE   0
#define    BEG_UPDATE   1
```

```
#define    PEN_INK       BLACK
#define    PEN_ERASER    WHITE

#define    PEN_FINE      1
#define    PEN_MEDIUM    5
#define    PEN_BROAD     9

#define    X_FWD         0x0100        /* extended object types       */
#define    X_BAK         0x0200        /* used with scrolling         */
#define    X_SEL         0x0300        /* selectors                   */
#define    N_COLORS      15L

#define    YSCALE(x) UMUL_DIV(x, scrn_xsize, scrn_ysize)

#define    TE_TXTLEN(x) (x + 24)
#define    BI_PDATA(x)    (x)
#define    BI_WB(x)       (x + 4)
#define    BI_HL(x)       (x + 6)

/*------------------------------*/
/*       SURE VARIABLES         */
/*------------------------------*/

extern     WORD   TIME;                /* mission time                */
extern     WORD   POINTS;              /* number of points plotted    */
extern     LONG   PRUNE;               /* deactivate th prunning      */
extern     WORD   WARNDIG;             /* set ub acc. to two digits   */
extern     WORD   TRUNC;               /* set loop traversals at 3    */
extern     WORD   LBFACT;              /* sets the Ki and Kj = 20     */
extern     WORD   LIST;                /* sets the basic output level */
double     speclow = -1;               /* variable range value        */
double     spechigh = -1;              /* variable range value        */
BOOLEAN    erun;                       /* runtime errors flag         */
BOOLEAN    nonlinear=FALSE;            /* nonlinear flag              */
double     specval;                    /* special variable value      */
double     ubfail;                     /* prob of system failure      */
double     lbfail;                     /* lower bound of failure      */
WORD       ydata = 1;                  /* Current data line           */
WORD       runno=0;                    /* run number                  */
WORD       pathcount = 0;             /* number of paths             */
WORD       cnttrunc = 0;              /* number of loops truncated   */
WORD       cntprunE = 0;              /* number of paths pruned      */
WORD       bigst = 1;                 /* largest state entered so far */
double     e_of_t;                    /* E(T) computation result     */
double     e_of_t_del;                /* E(T-*) computation result   */
BOOLEAN    et_bad = FALSE;            /* E(T) comp. is inaccurate    */
BOOLEAN    rec_slow = FALSE;          /* recovery too slow           */
BOOLEAN    std_big = FALSE;           /* Rec Std. too big            */
BOOLEAN    rate_big = FALSE;          /* Exp. rate too fast          */
WORD       i;
```

```
double     pialpha;                    /* product of the alphas        */
double     lowbf,lowbg;                /* intermediate results         */
WORD       num_states = 1;             /* number of states entered     */


/*    External Functions       */
/*-----------------------------*/

EXTERN LONG    dos_alloc();
/*-----------------------------*/
/*     Global Data Structures  */
/*-----------------------------*/

STATE_LOC   state_array[MAXSTATE];
/*********************************************************************/
/*********************************************************************/
/****                                                            ****/
/****          Data Structures                                   ****/
/****                                                            ****/
/*********************************************************************/
/*********************************************************************/

/*-----------------------------*/
/*    Extrnl Data Structures   */
/*-----------------------------*/

EXTERN    UWORD   DOS_ERR;
EXTERN    LONG    drawaddr;

/*-----------------------------*/
/*     Global Data Structures  */
/*-----------------------------*/

GLOBAL WORD   contrl[11];             /* control inputs               */
GLOBAL WORD   intin[80];              /* max string length            */
GLOBAL WORD   ptsin[256];             /* polygon fill points          */
GLOBAL WORD   intout[45];             /* open workstation output      */
GLOBAL WORD   ptsout[12];


/*-----------------------------*/
/*    Local Data Structures    */
/*-----------------------------*/

WORD    gl_wchar;                     /* character width              */
WORD    gl_hchar;                     /* character height             */
WORD    gl_wbox;                      /* box (cell) width             */
WORD    gl_hbox;                      /* box (cell) height            */
WORD    gl_hspace;                    /* Ht. of space between lines   */
WORD    gem_handle;                   /* GEM vdi handle               */
WORD    vdi_handle;                   /* SURE vdi handle              */
```

```
WORD    work_out[57];                   /* open virt workstation values*/
WORD    cur_fit = FULLITEM;             /* type of trnsfmin effect     */


GRECT   scrn_area;

GRECT   curr_work;                     /* work area of current window   */
GRECT   pict_work;                     /* work area of 'picture' window */
GRECT   data_work;
GRECT   pict_rect;                     /* work area of 'picture' window */
GRECT   data_rect;

GRECT   curr_undo;
GRECT   pict_undo;
GRECT   data_undo;

GRECT   pict_prev_undo;                 /* save area for full/unfulling */
GRECT   data_prev_undo;

WORD    gl_rmsg[8];                     /* message buffer               */
LONG    ad_rmsg;                        /* LONG pointer to message bfr  */
LONG    gl_menu;                        /* menu tree address            */
WORD    gl_apid;                        /* application ID               */
WORD    gl_xfull;                       /* full window 'x'              */
WORD    gl_yfull;                       /* full window 'y'              */
WORD    gl_wfull;                       /* full window 'w'              */
WORD    gl_hfull;                       /* full window 'h'              */
WORD    scrn_width;                     /* screen width in pixels       */
WORD    scrn_height;                    /* screen height in pixels      */
WORD    scrn_planes;                    /* number of color planes       */
WORD    scrn_xsize;                     /* width of one pixel           */
WORD    scrn_ysize;                     /* height of one pixel          */
UWORD   m_out = FALSE;                  /* mouse in/out of window flag  */
WORD    ev_which;                       /* event multi return state(s)  */
UWORD   mousex, mousey;                 /* mouse x,y position           */
UWORD   bstate, bclicks;                /* button state, & # of clicks  */
UWORD   kstate, kreturn;                /* key state and keyboard char  */

MFDB    curr_mfdb;
MFDB    pict_mfdb;
MFDB    data_mfdb;
MFDB    scrn_mfdb;

LONG    size_pict_mfdb;
LONG    size_data_mfdb;
LONG    loc_pict_mfdb;
LONG    loc_data_mfdb;

WORD    curr_hndl;
WORD    pict_hndl;                      /* SURE window handle           */
WORD    data_hndl;                      /* SURE data window handle      */
```

```
WORD    SURE_shade = PEN_INK;           /* SURE current pen shade       */
WORD    pen_shade = PEN_INK;            /* saved pen shade              */
WORD    SURE_pen = 5;                   /* SURE current pen width       */
WORD    radius = 16;                    /* radius of state              */
WORD    SURE_height = 4;                /* SURE current char height     */
WORD    char_fine;                      /* character height for fine    */
WORD    char_medium;                    /* character height for medium  */
WORD    char_broad;                     /* character height for broad   */
WORD    monumber = 5;                   /* mouse form number            */
LONG    mofaddr = 0x0L;                 /* mouse form address           */
WORD    file_handle;                    /* file handle -> pict ld/sv    */
BYTE    file_name[64] = "";             /* current pict file name       */
BOOLEAN key_input;                      /* key inputting state          */
WORD    key_xbeg;                       /* x posit for line beginning   */
WORD    key_ybeg;                       /* y posit for line beginning   */
WORD    key_xcurr;                      /* current x position           */
WORD    key_ycurr;                      /* current y position           */
                                        /* SURE window title            */
BYTE    *wdw_title = " SURE PICTURE WINDOW ";
BYTE    *wdw_tdata = " SURE DATA WINDOW ";

WORD    usercolor[2] = {1, 0};
MFDB    userbrush_mfdb;
USERBLK brushub[6];
LONG    color_sel[N_COLORS+1] = {       /* data for scrolling           */
    N_COLORS,                           /* color bar                    */
    0x31FF1071L,
    0x32FF1072L,
    0x33FF1073L,
    0x34FF1074L,
    0x35FF1075L,
    0x36FF1076L,
    0x37FF1077L,
    0x38FF1078L,
    0x39FF1079L,
    0x41FF107AL,
    0x42FF107BL,
    0x43FF107CL,
    0x44FF107DL,
    0x45FF107EL,
    0x46FF107FL};

/*------------------------------*/
/*    Mouse Data Structures     */
/*------------------------------*/

WORD    erase_broad[37] =               /* mouse form for broad eraser */
{
    7, 7, 1, 0, 1,
    0x0000, 0x0000, 0x0000, 0x0000,     /* mask for broad erase         */
    0x0000, 0x1ff0, 0x1ff0, 0x1ff0,
```

```
    0x1ff0, 0x1ff0, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000,     /* data for broad erase      */
    0x7ffc, 0x600c, 0x600c, 0x600c,
    0x600c, 0x600c, 0x7ffc, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000
};
WORD    erase_medium[37] =              /* mouse form for medium eraser*/
{
    7, 7, 1, 0, 1,
    0x0000, 0x0000, 0x0000, 0x0000,     /* mask for medium erase     */
    0x0000, 0x0000, 0x07c0, 0x07c0,
    0x07c0, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000,     /* data for medium erase     */
    0x0000, 0x1ff0, 0x1830, 0x1830,
    0x1830, 0x1ff0, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000
};
WORD   erase_fine[37] =                 /* mouse form for fine eraser  */
{
    7, 7, 1, 0, 1,
    0x0000, 0x0000, 0x0000, 0x0000,     /* mask for fine erase       */
    0x0000, 0x0000, 0x0000, 0x0100,
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000,     /* data for fine erase       */
    0x0000, 0x0000, 0x07c0, 0x06c0,
    0x07c0, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000
};



/************************************************************************/
/************************************************************************/
/****                                                               ****/
/****          Local Procedures                                     ****/
/****                                                               ****/
/************************************************************************/
/************************************************************************/


/*------------------------------*/
/*      min                     */
/*------------------------------*/
WORD
min(a, b)                               /* return min of two values    */
WORD    a, b;
{
    return( (a < b) ? a : b );
```

```
}


/*------------------------------*/
/*      max                     */
/*------------------------------*/
WORD
max(a, b)                            /* return max of two values    */
WORD   a, b;
{
   return( (a > b) ? a : b );
}



/*------------------------------*/
/*      reverse                 */
/*------------------------------*/
reverse(s)                           /* reverse string s in place   */
BYTE   s[];
{
   WORD    c,i,j;
   for (i=0, j=strlen(s)-1; i < j; i++,j--)
   {
       c = s[i];
       s[i] = s[j];
       s[j] = c;
   }
}
/*  reverse   */

/*------------------------------*/
/*      itoa                    */
/*------------------------------*/
itoa(n, s)                           /* convert n to characters in s*/
BYTE    s[];
WORD    n;
{
   WORD   i, sign;
   if ((sign = n) < 0)               /* record sign                 */
       n = -n;                       /* make n positive             */
   i = 0;
   do {                              /* generate digits in reverse  */
       s[i++] = n % 10 + '0';        /* order, get next digit       */
       } while ((n /= 10) > 0);      /* delete it                   */
   if (sign < 0)
       s[i++] = '-';
   s[i] = '\0';
   reverse(s);
}
/* itoa */
```

```
/*-------------------------------*/
/*      atof(s)                  */
/*-------------------------------*/
double
atof(s)                         /* Convert string s to double  */
BYTE   s[];
{
    double   val, power;
    WORD     i, sign;

    for ( i=0; s[i]==' ' || s[i]=='\n' || s[i]=='\t'; i++)
         ;                      /* Skip white space            */
    sign = 1;
    if (s[i] == '+' || s[i] == '-')    /* Sign                 */
         sign = (s[i++]=='+') ? 1 : -1;
    for (val = 0; s[i] >= '0' && s[i] <= '9'; i++)
         val = 10 * val + s[i] - '0';
    if (s[i] == '.')
         i++;
    for (power = 1; s[i] >= '0' && s[i] <= '9'; i++)
         {
              val = 10 * val + s[i] - '0';
              power *= 10;
         }
    return(sign * val / power);
} /* atof(s) */



/*------------------------------*/
/*      string_addr             */
/*------------------------------*/
LONG
string_addr(which)              /* returns a tedinfo LONG string addr */
WORD    which;
{
    LONG    where;

    rsrc_gaddr(R_STRING, which, &where);
    return (where);
}
```

```
/*------------------------------*/
/*      rc_equal                */
/*------------------------------*/
WORD
rc_equal(p1, p2)                    /* tests for two rectangles equal */
GRECT   *p1, *p2;
{
    if ((p1->g_x != p2->g_x) ||
        (p1->g_y != p2->g_y) ||
        (p1->g_w != p2->g_w) ||
        (p1->g_h != p2->g_h))
            return(FALSE);
    return(TRUE);
}



/*------------------------------*/
/*      rc_copy                 */
/*------------------------------*/
VOID
rc_copy(psbox, pdbox)         /* copy source to destination rectangle */
GRECT   *psbox;
GRECT   *pdbox;
{
    pdbox->g_x = psbox->g_x;
    pdbox->g_y = psbox->g_y;
    pdbox->g_w = psbox->g_w;
    pdbox->g_h = psbox->g_h;
}


/*------------------------------*/
/*      rc_intersect            */
/*------------------------------*/
WORD
rc_intersect(p1, p2)            /* compute intersect of two rectangles*/
GRECT   *p1, *p2;
{
    WORD    tx, ty, tw, th;

    tw = min(p2->g_x + p2->g_w, p1->g_x + p1->g_w);
    th = min(p2->g_y + p2->g_h, p1->g_y + p1->g_h);
    tx = max(p2->g_x, p1->g_x);
    ty = max(p2->g_y, p1->g_y);
    p2->g_x = tx;
    p2->g_y = ty;
    p2->g_w = tw - tx;
    p2->g_h = th - ty;
    return( (tw > tx) && (th > ty) );
}
```

```
/*------------------------------*/
/*      inside                  */
/*------------------------------*/
UWORD
inside(x, y, pt)               /* determine if x,y is in rectangle   */
UWORD    x, y;
GRECT    *pt;
{
    if ( (x >= pt->g_x) && (y >= pt->g_y) &&
         (x < pt->g_x + pt->g_w) && (y < pt->g_y + pt->g_h) )
         return(TRUE);
    else
         return(FALSE);
} /* inside */


/*------------------------------*/
/*      grect_to_array          */
/*------------------------------*/
VOID
grect_to_array(area, array)    /* convert x,y,w,h to upr lt x,y and  */
GRECT    *area;                /* lwr rt x,y                         */
WORD     *array;
{
    *array++ = area->g_x;
    *array++ = area->g_y;
    *array++ = area->g_x + area->g_w - 1;
    *array = area->g_y + area->g_h - 1;
}


/*------------------------------*/
/*      rast_op                 */
/*------------------------------*/
VOID
rast_op(mode, s_area, s_mfdb, d_area, d_mfdb) /* bit block level trns */
WORD    mode;
GRECT   *s_area, *d_area;
MFDB    *s_mfdb, *d_mfdb;
{
    WORD   pxy[8];

    grect_to_array(s_area, pxy);
    grect_to_array(d_area, &pxy[4]);
    vro_cpyfm(vdi_handle, mode, pxy, s_mfdb, d_mfdb);
}
```

```
/*------------------------------*/
/*      vdi_fix                 */
/*------------------------------*/
VOID
vdi_fix(pfd, theaddr, wb, h)
    MFDB   *pfd;
    LONG   theaddr;
    WORD   wb, h;
{
    pfd->fww = wb >> 1;
    pfd->fwp = wb << 3;
    pfd->fh = h;
    pfd->np = 1;
    pfd->mp = theaddr;
}

/*------------------------------*/
/*      vdi_trans               */
/*------------------------------*/
WORD
vdi_trans(saddr, swb, daddr, dwb, h)
    LONG   saddr;
    WORD   swb;
    LONG   daddr;
    WORD   dwb;
    WORD   h;
{
    MFDB   src, dst;

    vdi_fix(&src, saddr, swb, h);
    src.ff = TRUE;

    vdi_fix(&dst, daddr, dwb, h);
    dst.ff = FALSE;
    vr_trnfm(vdi_handle, &src, &dst );
}

/*------------------------------*/
/*      trans_gimage            */
/*------------------------------*/
VOID
trans_gimage(tree, obj)
    LONG    tree;
    WORD    obj;
{
    LONG    taddr, obspec;
    WORD    wb, hl;

    obspec = LLGET(OB_SPEC(obj));
    taddr = LLGET(BI_PDATA(obspec));
    wb = LWGET(BI_WB(obspec));
```

```
    hl = LWGET(BI_HL(obspec));
    vdi_trans(taddr, wb, taddr, wb, hl);
}

/*------------------------------*/
/*      do_open                 */
/*------------------------------*/
VOID
do_open(wh, org_x, org_y, x, y, w, h) /* grow and open specified wdw  */
WORD    wh;
WORD    org_x, org_y;
WORD    x, y, w, h;
{
    graf_growbox(org_x, org_y, 21, 21, x, y, w, h);
    wind_open(wh, x, y, w, h);
}

/*------------------------------*/
/*      do_close                */
/*------------------------------*/
VOID
do_close(wh, org_x, org_y)       /* close and shrink specified window */
WORD    wh;
WORD    org_x, org_y;
{
    WORD    x, y, w, h;

    wind_get(wh, WF_CXYWH, &x, &y, &w, &h);
    wind_close(wh);
    graf_shrinkbox(org_x, org_y, 21, 21, x, y, w, h);
}

/*------------------------------*/
/*      set_clip                */
/*------------------------------*/
VOID
set_clip(clip_flag, s_area)          /* set clip to specified area  */
WORD    clip_flag;
GRECT   *s_area;
{
    WORD    pxy[4];

    grect_to_array(s_area, pxy);
    vs_clip(vdi_handle, clip_flag, pxy);
}
```

```
/*------------------------------*/
/*      draw_rect               */
/*------------------------------*/
VOID
draw_rect(area)
GRECT   *area;
{
    WORD    pxy[10];

    pxy[0] = area->g_x;
    pxy[1] = area->g_y;
    pxy[2] = area->g_x + area->g_w - 1;
    pxy[3] = area->g_y + area->g_h - 1;
    pxy[4] = pxy[2];
    pxy[5] = pxy[3];
    pxy[3] = pxy[1];
    pxy[6] = pxy[0];
    pxy[7] = pxy[5];
    pxy[8] = pxy[0];
    pxy[9] = pxy[1];
    v_pline(vdi_handle, 5, pxy);
}


/*------------------------------*/
/*      align_x                 */
/*------------------------------*/
WORD
align_x(x)              /* forces word alignment for column position */
WORD    x;              /*   rounding to nearest word                */
{
    return((x & 0xfff0) + ((x & 0x000c) ? 0x0010 : 0));
}


/**********************************************************************/
/**********************************************************************/
/****                                                             ****/
/****         Advanced Dialog Handling                            ****/
/****                                                             ****/
/**********************************************************************/
/**********************************************************************/


/*------------------------------*/
/*   set_select                 */
/*------------------------------*/
VOID
set_select(tree, obj, init_no, bind, arry)
LONG   tree, bind[], arry[];
WORD   obj, init_no;
```

```
{
    WORDn, nobj, cobj, count;

    indir_obj(tree, obj);
    bind[0] = LLGET(OB_SPEC(obj));
    LLSET(OB_SPEC(obj), ADDR(bind));
    bind[1] = ADDR(arry);

    n = (WORD) arry[0];
    count = 0;
    for (cobj = LWGET(OB_HEAD(obj)); cobj != obj;
    cobj = LWGET(OB_NEXT(cobj)))
    {
         indir_obj(tree, cobj);
         LLSET(OB_SPEC(cobj), ADDR( &arry[count + 1] ));
         count = (count + 1) % n;
    }

    nobj = LWGET(OB_NEXT(obj));
    indir_obj(tree, nobj);
    LLSET(OB_SPEC(nobj), ADDR( &arry[1 + init_no % n] ));
}

/*------------------------------*/
/*       get_select             */
/*------------------------------*/
WORD
get_select(tree, obj)
LONG    tree;
WORD    obj;
{
    WORD   nobj, cobj;
    LONG   bind, arry, temp;

    bind = LLGET(OB_SPEC(obj));
    dir_obj(tree, obj);
    LLSET(OB_SPEC(obj), LLGET(bind));
    arry = LLGET(bind + sizeof(LONG) );

    for (cobj = LWGET(OB_HEAD(obj)); cobj != obj;
    cobj = LWGET(OB_NEXT(cobj)))
    {
         dir_obj(tree, cobj);
         LLSET(OB_SPEC(cobj), LLGET(LLGET(OB_SPEC(cobj))));
    }

    nobj = LWGET(OB_NEXT(obj));
    dir_obj(tree, nobj);
    temp = LLGET(OB_SPEC(nobj));
    LLSET(OB_SPEC(nobj), LLGET(temp));
    return (WORD) (temp - arry) / sizeof(LONG) - 1;
```

```
}

/*------------------------------*/
/*      move_do                 */
/*------------------------------*/
VOID
move_do(tree, obj, inc)
LONG    tree;
WORD    obj, inc;
{
    WORD   cobj;
    LONG   n, bind, arry, limit, obspec;

    obj = get_parent(tree, obj);
    obj = LWGET(OB_NEXT(obj));
    bind = LLGET(OB_SPEC(obj));
    arry = LLGET(bind + sizeof(LONG));
    n = LLGET(arry) * sizeof(LONG);
    limit = arry + n;

    for (cobj = LWGET(OB_HEAD(obj)); cobj != obj;
    cobj = LWGET(OB_NEXT(cobj)))
    {
         obspec = LLGET(OB_SPEC(cobj));
         obspec += inc * sizeof(LONG);
         while (obspec <= arry || obspec > limit)
              obspec += n * ((obspec > limit)? -1: 1);
         LLSET(OB_SPEC(cobj), obspec);
    }

    redraw_do(tree, obj);
}

/*------------------------------*/
/*      redraw_do               */
/*------------------------------*/
VOID
redraw_do(tree, obj)
LONG   tree;
WORD   obj;
{
    GRECT       o;

    objc_xywh(tree, obj, &o);
    o.g_x -= 3; o.g_y -= 3; o.g_w += 6; o.g_h += 6;
    objc_draw(tree, ROOT, MAX_DEPTH, o.g_x, o.g_y, o.g_w, o.g_h);
}
```

```
/*------------------------------*/
/*      xtend_do                */
/*------------------------------*/
WORD
xtend_do(tree, obj, xtype)
LONG    tree;
WORD    obj, xtype;
{
    LONG   obspec;

    switch (xtype) {
         case X_SEL:
              obspec = LLGET(OB_SPEC(obj));
              obj = get_parent(tree, obj);
              obj = LWGET(OB_NEXT(obj));
              LLSET(OB_SPEC(obj), obspec);
              redraw_do(tree, obj);
              break;
         case X_FWD:
              move_do(tree, obj, 1);
              redraw_do(tree, obj);
              break;
         case X_BAK:
              move_do(tree, obj, -1);
              redraw_do(tree, obj);
              break;
         default:
              break;
    }
    return(FALSE);
}

/*------------------------------*/
/*      hndl_dial               */
/*------------------------------*/
WORD
hndl_dial(tree, def, x, y, w, h)
LONG    tree;
WORD    def;
WORD    x, y, w, h;
{
    WORD    xdial, ydial, wdial, hdial, exitobj;
    WORD    xtype,i,j;

    form_center(tree, &xdial, &ydial, &wdial, &hdial);
    form_dial(0, x, y, w, h, xdial, ydial, wdial, hdial);
    form_dial(1, x, y, w, h, xdial, ydial, wdial, hdial);
    objc_draw(tree, ROOT, MAX_DEPTH, xdial, ydial, wdial, hdial);
```

```
    FOREVER
    {
    exitobj = form_do(tree, def) & 0x7FFF;
    xtype = LWGET(OB_TYPE(exitobj)) & 0xFF00;
    if (!xtype)
            break;
        if (xtend_do(tree, exitobj, xtype))
            break;
    }
    form_dial(2, x, y, w, h, xdial, ydial, wdial, hdial);
    form_dial(3, x, y, w, h, xdial, ydial, wdial, hdial);
    return (exitobj);
}

/*------------------------------*/
/*      dr_code                 */
/*------------------------------*/
WORD
dr_code(pparms)
LONG    pparms;
{
    PARMBLK    pb;
    WORD       pxy[10], hl, wb;
    LONG       taddr;

    LBCOPY(ADDR(&pb), pparms, sizeof(PARMBLK));
    set_clip(TRUE, (GRECT *) &pb.pb_xc);

    taddr = pb.pb_parm;
    userbrush_mfdb.mp = LLGET(BI_PDATA(taddr));
    hl = LWGET(BI_HL(taddr));
    wb = LWGET(BI_WB(taddr));
    userbrush_mfdb.fwp = wb << 3;
    userbrush_mfdb.fww = wb >> 1;
    userbrush_mfdb.fh = hl;
    userbrush_mfdb.np = 1;
    userbrush_mfdb.ff = 0;

    pxy[0] = pxy[1] = 0;
    pxy[2] = (wb << 3) - 1;
    pxy[3] = hl - 1;
    pxy[4] = pb.pb_x;
    pxy[5] = pb.pb_y;
    pxy[6] = pxy[4] + pb.pb_w - 1;
    pxy[7] = pxy[5] + pb.pb_h - 1;

    vrt_cpyfm(vdi_handle, 2, pxy,&userbrush_mfdb,&scrn_mfdb, usercolor);
```

```
    if((pb.pb_currstate!=pb.pb_prevstate)||(pb.pb_currstate&SELECTED))
    {
         if (pb.pb_currstate & SELECTED)
              vsl_color(vdi_handle,1);
         else
              vsl_color(vdi_handle,0);
         vsl_width(vdi_handle, 1);
         vsl_type(vdi_handle, FIS_SOLID);

         pb.pb_x--;
         pb.pb_y--;
         pb.pb_w++;
         pb.pb_h++;
         draw_rect((GRECT *) &pb.pb_x);
     }
     set_clip(TRUE, &pict_work);
     return (0);
}

/*************************************************************************/
/*************************************************************************/
/****                                                               ****/
/****         Work Area Management                                  ****/
/****                                                               ****/
/*************************************************************************/
/*************************************************************************/


/*------------------------------*/
/*       set_work               */
/*------------------------------*/


VOID
set_pict_work(slider_update)
BOOLEAN   slider_update;            /*edges, and updt sliders if req'd */
{
    WORD   i;

    wind_get(pict_hndl, WF_WXYWH,
    &pict_work.g_x,&pict_work.g_y,&pict_work.g_w,&pict_work.g_h);

    pict_undo.g_w = pict_work.g_w;
    pict_undo.g_h = pict_work.g_h;
/**/                                 /* clamp work area to page edges */
    pict_undo.g_x = align_x(pict_undo.g_x);
    if ((i = pict_mfdb.fwp - (pict_undo.g_x + pict_undo.g_w)) < 0)
         pict_undo.g_x += i;
    if ((i = pict_mfdb.fh - (pict_undo.g_y + pict_undo.g_h)) < 0)
         pict_undo.g_y += i;
```

```
    if (slider_update )
    {
        wind_set(pict_hndl, WF_HSLIDE, UMUL_DIV(pict_undo.g_x, 1000,
            pict_mfdb.fwp - pict_undo.g_w), 0, 0, 0);
        wind_set(pict_hndl, WF_VSLIDE, UMUL_DIV(pict_undo.g_y, 1000,
            pict_mfdb.fh - pict_undo.g_h), 0, 0, 0);
        wind_set(pict_hndl, WF_HSLSIZ, UMUL_DIV(pict_work.g_w, 1000,
            pict_mfdb.fwp), 0, 0, 0);
        wind_set(pict_hndl, WF_VSLSIZ, UMUL_DIV(pict_work.g_h, 1000,
            pict_mfdb.fh), 0, 0, 0);

    }

                          /* only use portion of work_larea on screen */
    rc_intersect(&scrn_area, &pict_work);
    pict_undo.g_w = pict_work.g_w;
    pict_undo.g_h = pict_work.g_h;
}


VOID
set_data_work(slider_update)
BOOLEAN   slider_update;          /*edges, and updt sliders if req'd */
{
    WORD   i;

    wind_get(data_hndl, WF_WXYWH,
    &data_work.g_x,&data_work.g_y,&data_work.g_w,&data_work.g_h);

    data_undo.g_w = data_work.g_w;
    data_undo.g_h = data_work.g_h;
/**/                                  /* clamp work area to page edges */
    data_undo.g_x = align_x(data_undo.g_x);
    if ((i = data_mfdb.fwp - (data_undo.g_x + data_undo.g_w)) < 0)
        data_undo.g_x += i;
    if ((i = data_mfdb.fh - (data_undo.g_y + data_undo.g_h)) < 0)
        data_undo.g_y += i;

    if (slider_update )
    {
        wind_set(data_hndl, WF_HSLIDE, UMUL_DIV(data_undo.g_x, 1000,
            data_mfdb.fwp - data_undo.g_w), 0, 0, 0);
        wind_set(data_hndl, WF_VSLIDE, UMUL_DIV(data_undo.g_y, 1000,
            data_mfdb.fh - data_undo.g_h), 0, 0, 0);
        wind_set(data_hndl, WF_HSLSIZ, UMUL_DIV(data_work.g_w, 1000,
            data_mfdb.fwp), 0, 0, 0);
        wind_set(data_hndl, WF_VSLSIZ, UMUL_DIV(data_work.g_h, 1000,
            data_mfdb.fh), 0, 0, 0);

    }
```

```
                        /* only use portion of work_iarea on screen */
    rc_intersect(&scrn_area, &data_work);
    data_undo.g_w = data_work.g_w;
    data_undo.g_h = data_work.g_h;
}




/*------------------------------*/
/*       save_work              */
/*------------------------------*/

VOID
save_pict_work()                        /* copy pict_work to pict_mfdb */
{
    GRECT   tmp_area;

    rc_copy(&pict_work,&tmp_area);
    rc_intersect(&scrn_area,&tmp_area);
    graf_mouse(M_OFF, 0x0L);
    rast_op(3, &tmp_area, &scrn_mfdb, &pict_undo, &pict_mfdb);
    graf_mouse(M_ON, 0x0L);
}


VOID
save_data_work()                        /* copy data_work to data_mfdb */
{
    GRECT   tmp_area;

    rc_copy(&data_work,&tmp_area);
    rc_intersect(&scrn_area,&tmp_area);
    graf_mouse(M_OFF, 0x0L);
    rast_op(3, &tmp_area, &scrn_mfdb, &data_undo, &data_mfdb);
    graf_mouse(M_ON, 0x0L);
}



/*------------------------------*/
/*       restore_work           */
/*------------------------------*/

VOID
rst_pict_work()                   /* restore pict_work from pict_mfdb */
{
    GRECT    tmp_area;

    rc_copy(&pict_work,&tmp_area);
```

```
    rc_intersect(&scrn_area,&tmp_area);
    graf_mouse(M_OFF, 0x0L);
    rast_op(3, &pict_undo, &pict_mfdb, &tmp_area, &scrn_mfdb);
    graf_mouse(M_ON, 0x0L);
}


VOID
rst_data_work()                    /* restore data_work from data_mfdb */
{
    GRECT    tmp_area;

    rc_copy(&data_work,&tmp_area);
    rc_intersect(&scrn_area,&tmp_area);
    graf_mouse(M_OFF, 0x0L);
    rast_op(3, &data_undo, &data_mfdb, &tmp_area, &scrn_mfdb);
    graf_mouse(M_ON, 0x0L);
}


/***********************************************************************/
/***********************************************************************/
/****                                                              ****/
/****          Object Tree Manipulation                            ****/
/****                                                              ****/
/***********************************************************************/
/***********************************************************************/


/*------------------------------*/
/*       do_obj                 */
/*------------------------------*/
VOID
do_obj(tree, which, bit)        /* set specified bit in object state  */
LONG    tree;
WORD    which, bit;
{
    WORD    state;

    state = LWGET(OB_STATE(which));
    LWSET(OB_STATE(which), state | bit);
}


/*------------------------------*/
/*       undo_obj               */
/*------------------------------*/
VOID
undo_obj(tree, which, bit)    /* clear specified bit in object state  */
LONG    tree;
WORD    which, bit;
```

```
{
    WORD    state;

    state = LWGET(OB_STATE(which));
    LWSET(OB_STATE(which), state & ~bit);
}

/*------------------------------*/
/*      sel_obj                 */
/*------------------------------*/
VOID
sel_obj(tree, which)          /* turn on selected bit of spcfd object */
LONG    tree;
WORD    which;
{
    do_obj(tree, which, SELECTED);
}

/*------------------------------*/
/*      desel_obj               */
/*------------------------------*/
VOID
desel_obj(tree, which)        /* turn off selected bit of spcfd object*/
LONG    tree;
WORD    which;
{
    undo_obj(tree, which, SELECTED);
}

/*------------------------------*/
/*      enab_menu               */
/*------------------------------*/
VOID
enab_menu(which)                       /* enable specified menu item  */
WORD    which;
{
    undo_obj(gl_menu, which, DISABLED);
}

/*------------------------------*/
/*      unflag_obj              */
/*------------------------------*/
VOID
unflag_obj(tree, which, bit)
LONG    tree;
WORD    which, bit;
{
    WORD    flags;

    flags = LWGET(OB_FLAGS(which));
    LWSET(OB_FLAGS(which), flags & ~bit);
```

```
}

/*------------------------------*/
/*      flag_obj                */
/*------------------------------*/
VOID
flag_obj(tree, which, bit)
LONG    tree;
WORD    which, bit;
{
    WORD   flags;

    flags = LWGET(OB_FLAGS(which));
    LWSET(OB_FLAGS(which), flags | bit);
}

/*------------------------------*/
/*      indir_obj               */
/*------------------------------*/
VOID
indir_obj(tree, which)
LONG   tree;
WORD   which;
{
    flag_obj(tree, which, INDIRECT);
}

/*------------------------------*/
/*      dir_obj                 */
/*------------------------------*/
VOID
dir_obj(tree, which)
LONG    tree;
WORD    which;
{
    unflag_obj(tree, which, INDIRECT);
}

/*------------------------------*/
/*      get_parent              */
/*------------------------------*/
/*
*   Routine that will find the parent of a given object.  The
*   idea is to walk to the end of our siblings and return
*   our parent.  If object is the root then return NIL as parent.
*/
WORD
get_parent(tree, obj)
LONG    tree;
WORD    obj;
{
```

```
    WORD   pobj;

    if (obj == NIL)
    return (NIL);
    pobj = LWGET(OB_NEXT(obj));
    if (pobj != NIL)
    {
         while( LWGET(OB_TAIL(pobj)) != obj )
         {
              obj = pobj;
              pobj = LWGET(OB_NEXT(obj));
         }
    }
    return(pobj);
}

/*------------------------------*/
/*      objc_xywh               */
/*------------------------------*/
VOID
objc_xywh(tree, obj, p)         /* get x,y,w,h for specified object   */
LONG    tree;
WORD    obj;
GRECT   *p;
{
    objc_offset(tree, obj, &p->g_x, &p->g_y);
    p->g_w = LWGET(OB_WIDTH(obj));
    p->g_h = LWGET(OB_HEIGHT(obj));
}


/***********************************************************************/
/***********************************************************************/
/****                                                              ****/
/****         File Path Name Functions                             ****/
/****                                                              ****/
/***********************************************************************/
/***********************************************************************/


/*------------------------------*/
/*   dial_name                  */
/*------------------------------*/
WORD
dial_name ( name )                     /* dialogue box input filename */
BYTE    *name;
{
    LONG   tree ;
    LONG   ted_addr ;
    BYTE   c ;
    WORD   i, j;
```

```
    GRECT  box;


    objc_xywh(gl_menu, SUREFILE, &box);
    rsrc_gaddr( R_TREE, SURESVAD, &tree) ;
    ted_addr = LLGET(OB_SPEC(SURENAME));
    LLSET( ted_addr, ADDR(name) ) ;
    LWSET( TE_TXTLEN(ted_addr),8);
    name[0] = '\0';
    if(hndl_dial(tree,SURENAME,box.g_x,box.g_y,box.g_w,box.g_h)==SURESOK)
    {
         i =
         j =  0;
         while (TRUE)
         {
             c = name[i++];
             if (!c)
                 break ;
             if ( (c != ' ') && (c != '_') )
                 name[j++] = c ;
         }
         if ( *name )
              strcpy( &name[j], ".DOO" ) ;
         desel_obj(tree, SURESOK);
         return (TRUE);
    }
    else
    {
         desel_obj(tree, SURESCNL);
         return (FALSE);
    }
}


/*------------------------------*/
/*       get_path               */
/*------------------------------*/
VOID
get_path(tmp_path, spec)                /* get directory path name     */
BYTE    *tmp_path, *spec;
{
    WORD    cur_drv;

    cur_drv = dos_gdrv();
    tmp_path[0] = cur_drv + 'A';
    tmp_path[1] = ':';
    tmp_path[2] = '\\';
    dos_gdir(cur_drv+1, ADDR(&tmp_path[3]));
    if (strlen(tmp_path) > 3)
         strcat(tmp_path, "\\");
    else
```

```
        tmp_path[2] = '\0';
    strcat(tmp_path, spec);
}


/*------------------------------*/
/*      add_file_name           */
/*------------------------------*/
VOID
add_file_name(dname, fname)  /* replace name at end of input file spec*/
BYTE    *dname, *fname;
{
    BYTE    c;
    WORD    ii;


    ii = strlen(dname);
    while (ii && ((c = dname[ii-1])  != '\\') && (c != ':')))
         ii--;
    dname[ii] = '\0';
    strcat(dname, fname);
}


/*------------------------------*/
/*      get_file                */
/*------------------------------*/
WORD
get_file(loop)                  /* use file selector to get input file*/
BOOLEAN    loop;
{
    WORD    fs_iexbutton;
    BYTE    fs_iinsel[13];

  while (TRUE)
    {
         get_path(file_name, "*.DOO");
         fs_iinsel[0] = '\0';

         fsel_input(ADDR(file_name), ADDR(fs_iinsel), &fs_iexbutton);
         if (fs_iexbutton)
         {
              add_file_name(file_name, fs_iinsel);
              file_handle = dos_open(ADDR(file_name),2);
              if (!loop || (loop && !DOS_ERR))
                   return(1);
         }
         else
              return (0);
    }
```

```
}   /* get_file */


/***************************************************************************/
/***************************************************************************/
/****                                                                  ****/
/****              Soft Cursor Support                                 ****/
/****                                                                  ****/
/***************************************************************************/
/***************************************************************************/


/*-------------------------------*/
/*      cursor                   */
/*-------------------------------*/
VOID
cursor(color)                         /* turn cursor on, color=BLACK */
WORD    color;                        /* or cursor off, color= WHITE */
{
    WORD    pxy[4];

    pxy[0] = key_xcurr + 1;
    pxy[1] = key_ycurr + gl_hspace;
    pxy[2] = key_xcurr + 1;
    pxy[3] = key_ycurr - gl_hbox;

    vsl_color(vdi_handle,color);
    vswr_mode(vdi_handle,MD_REPLACE);
    vsl_type (vdi_handle,FIS_SOLID);
    vsl_width (vdi_handle,PEN_FINE);
    graf_mouse(M_OFF, 0x0L);

    wind_get(pict_hndl,WF_TOP,&curr_hndl,0,0,0);


    if (curr_hndl == pict_hndl) set_clip(TRUE, &pict_work);
    if (curr_hndl == data_hndl) set_clip(TRUE, &data_work);
    v_pline(vdi_handle, 2, pxy);


    if (curr_hndl == pict_hndl ) set_clip(FALSE, &pict_work);
    if (curr_hndl == data_hndl ) set_clip(FALSE, &data_work);
    graf_mouse(M_ON, 0x0L);
}
```

```
/*------------------------------*/
/*      curs_on                 */
/*------------------------------*/
VOID
curs_on()                            /* turn 'soft' cursor 'on'     */
{
   cursor(pen_shade);
}

/*------------------------------*/
/*      curs_off                */
/*------------------------------*/
VOID
curs_off()                           /* turn 'soft' cursor 'off'    */
{
   cursor(PEN_ERASER);
}


/************************************************************************/
/************************************************************************/
/****                                                               ****/
/****             Menu Handling                                     ****/
/****                                                               ****/
/************************************************************************/
/************************************************************************/


/*------------------------------*/
/*    hndl_menu                 */
/*------------------------------*/       /* curr_hndl is known if
                                            called from hndl_msg   */
WORD
hndl_menu(title, item)
WORD    title, item;
{
    WORD    done;
    GRECT   box;
    LONG    tree;

    graf_mouse(ARROW, 0x0L);
    done = FALSE;
    switch (title) {
    case SUREDESK:                   /* menu selection name-DESK    */
         if (item == SUREINFO)
              do_about();
         break;

    case SUREFILE:                   /* menu selection name-FILE    */
         switch (item)
         {
```

```
     case SURELOAD:                 /* menu selection name-LOAD    */
          do_load(TRUE);
          break;
     case SURESAVE:                 /* menu selection name-SAVE    */
          do_save();
          break;
     case SURESVAS:                 /* menu selection name-SAVEAS  */
          do_svas();
          break;
     case SUREABAN:                 /* menu selection name-ABANDON */
          file_handle = dos_open(ADDR(file_name),2);
          do_load(FALSE);
          break;
     case SUREQUIT:                 /* menu selection name-QUIT    */
          done = TRUE;
          break;
     }

case SCROPTS:                       /* menu select name-SCR-OPTIONS*/
     switch (item)
     {
     case SUREPENS:                 /* menu select name-PENS/ERASER*/
          do_penselect();
          break;
     case SUREERAP:                 /* menu selection name-ERASE   */
          do_erase();
          break;
     case VIEWITEM:                 /* menu selection name-VIEW ...*/
          do_view();
          break;
     }
case SUREOPTS:                      /* menu name-SURE-OPTIONS      */
     switch (item)
     {
     case SUREPLOT:                 /* menu name-RELIABILITY PLOT  */
          do_plot();
          break;
     case SURECOMP:                 /* menu name-COMPUTE           */
          do_compute();
          break;
     case SUREHARD:                 /* menu name-HARDCOPY          */
          do_hardcopy();
          break;
     case SLOWTRAN:                 /* menu name-SLOW  TRANSITION  */
          do_slow_transition();
          break;
     case FASTTRAN:                 /* menu name-FAST TRANSITION   */
          do_fast_transition();
          break;
     case SURESTAT:                 /* menu name-ENTER STATES      */
          do_states();
```

```
            break;
        case SURECONS:                  /* menu name-ENTER CONSTANTS   */
            do_constants();
            break;
        }
    }
    menu_tnormal(gl_menu,title,TRUE);
    graf_mouse(monumber, mofaddr);
    return (done);
}


/*------------------------------*/
/*      do_about                */
/*------------------------------*/
VOID
do_about()                          /* display SURE Info...        */
{
    LONG   tree;
    GRECT  box;

    objc_xywh(gl_menu, SUREDESK, &box);
    rsrc_gaddr(R_TREE, SUREINFO, &tree);
    hndl_dial(tree, 0, box.g_x, box.g_y, box.g_w, box.g_h);
    desel_obj(tree, SUREOK);
}

/*-----------------------------*/
/*      do_view                */
/*-----------------------------*/
VOID
do_view()                           /* display zooming available   */
{
    LONG   tree;
    GRECT  box;

    set_clip(TRUE, &pict_undo);
    objc_xywh(gl_menu, SUREDESK, &box);
    rsrc_gaddr(R_TREE, FITDIAL, &tree);
    hndl_dial(tree, 0, box.g_x, box.g_y, box.g_w, box.g_h);
    desel_obj(tree, FITOK);
    for ( cur_fit = FULLITEM ;
          (!(LWGET( OB_STATE( cur_fit ) ) & SELECTED) ) ;
          cur_fit++ )
          ;
/*  set_xform() ;
    set_clip( TRUE, &pict_work_area ) ;
    rdr_mesg();*/
    set_clip(FALSE,&pict_undo);
}   /* do_view    */
```

```
/*------------------------------*/
/*   do_load                    */
/*------------------------------*/
                                      /* curr_hndl is known */
VOID
do_load(need_name)                    /* load  file      */
BOOLEAN    need_name;
{
if (!need_name || get_file(TRUE))
   {
   if (!DOS_ERR)
      {
         dos_read(file_handle, size_pict_mfdb,loc_pict_mfdb);
         dos_read(file_handle, size_data_mfdb,loc_data_mfdb);

      dos_close(file_handle);
      enab_menu(SURESAVE);
      enab_menu(SUREABAN);
      rst_pict_work();
      rst_data_work();
      }
   }
}


/*------------------------------*/
/*   do_save                    */
/*------------------------------*/
VOID
do_save()                        /* save current named SURE picture  */
{
wind_get(pict_hndl,WF_TOP,&curr_hndl,0,0,0);
if (*file_name)
   {
   file_handle = dos_open(ADDR(file_name),2);
   if (DOS_ERR)
      file_handle = dos_create(ADDR(file_name),0);
   else
      {
      if (form_alert(1, string_addr(SUREOVWR)) == 2)
         return;
      }
      dos_write(file_handle, size_pict_mfdb, loc_pict_mfdb);
      dos_write(file_handle, size_data_mfdb, loc_data_mfdb);
   enab_menu(SURESAVE);
   enab_menu(SUREABAN);
   dos_close(file_handle);
   }
}
```

```
/*------------------------------*/
/*       do_save_as             */
/*------------------------------*/
VOID
do_svas()                               /* save SURE picture as named  */
{
    BYTE   name[13];

    if (dial_name(&name[0]))
    {
         add_file_name(file_name, name);
         do_save();
    }
}

/*------------------------------*/
/*      set_pen                 */
/*------------------------------*/
VOID
set_pen(pen, height)
WORD   pen, height;
{
    SURE_pen = pen;
    SURE_height = height;
    monumber = 5;
    mofaddr = 0x0L;
}

/*------------------------------*/
/*   set_eraser                 */
/*------------------------------*/
VOID
set_eraser(pen, height, eraser)
WORD    pen, height;
BYTE    *eraser;
{
    SURE_pen = pen;
    SURE_height = height;
    SURE_shade = PEN_ERASER;
    monumber = 255;
    mofaddr = ADDR(eraser);
}

/*------------------------------*/
/*      set_color               */
/*------------------------------*/
VOID
set_color(tree, obj, color_num, bind)
LONG    tree, *bind;
WORD    obj, color_num;
{
```

```
    set_select(tree, obj, color_num - 1, bind, color_sel);
}

/*------------------------------*/
/*      get_color               */
/*------------------------------*/
WORD
get_color(tree, obj)
LONG   tree;
WORD   obj;
{
    return get_select(tree, obj) + 1;
}

/*------------------------------*/
/*      do_penselect            */
/*------------------------------*/
VOID
do_penselect()                  /* use dialogue box to input selection */
{                               /* of specified pen/eraser             */
    WORD    exit_obj, psel_obj, color;
    LONG    tree, bind[2];
    GRECT   box;

    set_clip(TRUE, &pict_undo);
    objc_xywh(gl_menu, SUREPENS, &box);
    rsrc_gaddr(R_TREE, SUREPEND, &tree);
    /**/                        /* first setup current selection state*/
    switch (SURE_pen) {
        case PEN_FINE:
            sel_obj(tree, (SURE_shade != PEN_ERASER)?
                SUREPFIN: SUREEFIN);
            radius = 10;
            break;
        case PEN_MEDIUM:
            sel_obj(tree, (SURE_shade != PEN_ERASER)?
                SUREPMED: SUREEMED);
            radius = 16;
            break;
        case PEN_BROAD:
            sel_obj(tree, (SURE_shade != PEN_ERASER)?
                SUREPBRD: SUREEBRD);
            radius = 20;
            break;
    }
    set_color(tree, DEMOPCLR, pen_shade, bind);
    /**/                                /* get dialogue box input      */
    exit_obj = hndl_dial(tree, 0, box.g_x, box.g_y, box.g_w, box.g_h);
    for (psel_obj = SUREPFIN; psel_obj <= SUREEBRD; psel_obj++)
```

```
        if (LWGET(OB_STATE(psel_obj)) & SELECTED)
        {
             desel_obj(tree, psel_obj);
             break;
        }
     color = get_color(tree, DEMOPCLR);

    if (exit_obj == SUREPSOK)
    {
        switch (psel_obj) {
             case SUREPFIN:
                  set_pen(PEN_FINE, char_fine);
                  SURE_shade = color;
                  break;
             case SUREPMED:
                  set_pen(PEN_MEDIUM, char_medium);
                  SURE_shade = color;
                  break;
             case SUREPBRD:
                  set_pen(PEN_BROAD, char_broad);
                  SURE_shade = color;
                  break;
             case SUREEFIN:
                  set_eraser(PEN_FINE, char_fine,
                       (BYTE *) erase_fine);
                  break;
             case SUREEMED:
                  set_eraser(PEN_MEDIUM, char_medium,
                       (BYTE *) erase_medium);
                  break;
             case SUREEBRD:
                  set_eraser(PEN_BROAD, char_broad,
                       (BYTE *) erase_broad);
                  break;
        }
        pen_shade = color;
        desel_obj(tree, SUREPSOK);
    }
    else
        desel_obj(tree, SURECNCL);
    set_clip(FALSE,&pict_undo);
}


/*------------------------------*/
/*      do_erase                */
/*------------------------------*/
VOID
do_erase()              /* clear the 'picture' and 'data' buffer (mfdb's)
                           and re-init. the screen                     */
{
```

```
GRECT tmp_area;

tmp_area.g_x = 0;
tmp_area.g_y = 0;
tmp_area.g_w = pict_mfdb.fwp;
tmp_area.g_h = pict_mfdb.fh;

rast_op(0,&tmp_area,&pict_mfdb,&tmp_area,&pict_mfdb);

tmp_area.g_w = data_mfdb.fwp;
tmp_area.g_h = data_mfdb.fh;

rast_op(0,&tmp_area,&data_mfdb,&tmp_area,&data_mfdb);
rst_pict_work();
rst_data_work();
init_data();
num_states = 1;
ydata = 1;
}



/************************************************************************/
/************************************************************************/
/****                                                               ****/
/****               Keyboard Handling                               ****/
/****                                                               ****/
/************************************************************************/
/************************************************************************/


/*------------------------------*/
/*      hndl_keyboard          */
/*------------------------------*/
WORD
hndl_keyboard()
{
    WORD    i;
    BYTE    str[2];
    GRECT   lttr, test;

    if ((str[0] = kreturn) == 0x03)
         return(TRUE);
    graf_mouse(M_OFF, 0x0L);
    if (!key_input)
    {
         vswr_mode(vdi_handle, MD_REPLACE);
         vst_color(vdi_handle, pen_shade);
         vst_height(vdi_handle, SURE_height,
              &gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
         gl_hspace = gl_hbox - gl_hchar;
```

```
	vst_alignment(vdi_handle, 0, 0, &i, &i);
	graf_mkstate(&key_xbeg, &key_ybeg, &i, &i);
	key_xcurr = ++key_xbeg-8;
	key_ycurr = --key_ybeg+4;
}
else
	curs_off();
str[1] = '\0';
if (str[0] == 0x1A)
{
	save_pict_work();
	graf_mouse(M_ON, 0x0L);
	return(key_input = FALSE);
}
else
if (str[0] == 0x0D)
	{
		key_ycurr += gl_hbox + gl_hspace;
		key_xcurr = key_xbeg;
	}
	else
		if (str[0] == 0x08)
		{
			if (key_input && (key_xcurr != key_xbeg))
			{
				for (i = 0; i < gl_wbox; i++)
				{
					key_xcurr--;
					curs_off();
				}
			}
		}
		else
			if ((str[0] >= ' ') && (str[0] <= 'z'))
			{
				lttr.g_x = key_xcurr;
				lttr.g_y = key_ycurr - gl_hbox;
				lttr.g_w = gl_wbox;
				lttr.g_h = gl_hbox;
				rc_copy(&lttr, &test);
				rc_intersect(&pict_work, &test);
				if (!rc_equal(&lttr, &test))
					{
					graf_mouse(M_ON, 0x0L);
					return (FALSE);
					}
				set_clip(TRUE, &pict_work);
				v_gtext(vdi_handle, key_xcurr,
					key_ycurr, str);
				set_clip(FALSE, &pict_work);
				key_xcurr += gl_wbox;
```

```
                }
    if (!key_input)
    {
         key_input = TRUE;
    }
    curs_on();
    graf_mouse(M_ON, 0x0L);
    return(FALSE);
}



/*************************************************************************/
/*************************************************************************/
/****                                                               ****/
/****              Message Handling                                 ****/
/****                                                               ****/
/*************************************************************************/
/*************************************************************************/


/*------------------------------*/
/*       hndl_msg               */
/*------------------------------*/
/*MLOCAL*/   BOOLEAN    hndl_msg()
{
    BOOLEAN  done;
    GRECT    work;

    done = FALSE;
    curr_hndl = wind_find(mousex,mousey);
    if ( curr_hndl == pict_hndl )
       {
       rc_copy(&pict_undo,&curr_undo);
       rc_copy(&pict_work,&curr_work);
       curr_mfdb.fwp = pict_mfdb.fwp;
       curr_mfdb.fh  = pict_mfdb.fh;
       }
    if ( curr_hndl == data_hndl )
       {
       rc_copy(&data_undo,&curr_undo);
       rc_copy(&data_work,&curr_work);
       curr_mfdb.fwp = data_mfdb.fwp;
       curr_mfdb.fh  = data_mfdb.fh;
       }
```

```
switch( gl_rmsg[0] )
{
case MN_SELECTED:
     done = hndl_menu(gl_rmsg[3],gl_rmsg[4]);
     break;

case WM_REDRAW:
     do_redraw(gl_rmsg[3], (GRECT *) &gl_rmsg[4]);
     break;

case WM_TOPPED:
     wind_set(gl_rmsg[3], WF_TOP, 0, 0, 0, 0);
     break;
case WM_CLOSED:
     done = TRUE;
     break;
case WM_FULLED:
     do_full(curr_hndl);
     break;
case WM_ARROWED:

     switch(gl_rmsg[4])
     {
     case WA_UPPAGE:
          curr_undo.g_y = max(curr_undo.g_y - curr_undo.g_h, 0);
          break;
     case WA_DNPAGE:
          curr_undo.g_y += curr_undo.g_h;
          break;
     case WA_UPLINE:
          curr_undo.g_y = max(curr_undo.g_y - YSCALE(16), 0);
          break;
     case WA_DNLINE:
          curr_undo.g_y += YSCALE(16);
          break;
     case WA_LFPAGE:
          curr_undo.g_x = max(curr_undo.g_x-curr_undo.g_w, 0);
          break;
     case WA_RTPAGE:
          curr_undo.g_x += curr_undo.g_w;
          break;
     case WA_LFLINE:
          curr_undo.g_x = max(curr_undo.g_x - 16, 0);
          break;
     case WA_RTLINE:
          curr_undo.g_x += 16;
          break;
     }
     if ( curr_hndl == pict_hndl )
        {
        rc_copy(&curr_undo,&pict_undo);
```

```
        set_pict_work(TRUE);
        rst_pict_work();
        }
     if ( curr_hndl == data_hndl )
        {
        rc_copy(&curr_undo,&data_undo);
        set_data_work(TRUE);
        rst_data_work();
        }
     break;

case WM_HSLID:
     curr_undo.g_x = align_x(UMUL_DIV(curr_mfdb.fwp - curr_undo.g_w,
     gl_rmsg[4], 1000));

     if ( curr_hndl == pict_hndl )
        {
        pict_undo.g_x = curr_undo.g_x;
        set_pict_work(TRUE);
        rst_pict_work();
        }
     if (  curr_hndl == data_hndl )
        {
        data_undo.g_x = curr_undo.g_x;
        set_data_work(TRUE);
        rst_data_work();
        }
     break;

case WM_VSLID:
     curr_undo.g_y = UMUL_DIV(curr_mfdb.fh - curr_undo.g_h,
     gl_rmsg[4],1000);

     if ( curr_hndl == pict_hndl )
        {
        pict_undo.g_y = curr_undo.g_y;
        set_pict_work(TRUE);
        rst_pict_work();
        }
     if (  curr_hndl == data_hndl )
        {
        data_undo.g_y = curr_undo.g_y;
        set_data_work(TRUE);
        rst_data_work();
        }

     break;
```

```
    case WM_SIZED:
         curr_hndl = wind_find(mousex,mousey);
         if (curr_hndl == pict_hndl)
         {
         wind_calc(1, 0x0fef, gl_rmsg[4], gl_rmsg[5], gl_rmsg[6],
              gl_rmsg[7],&pict_work.g_x,&pict_work.g_y,&pict_work.g_w,
              &pict_work.g_h);
         pict_work.g_x = align_x(pict_work.g_x);
         pict_work.g_w = align_x(pict_work.g_w);
         wind_calc(0, 0x0fef,pict_work.g_x,pict_work.g_y,pict_work.g_w,
         pict_work.g_h,&gl_rmsg[4],&gl_rmsg[5],&gl_rmsg[6],&gl_rmsg[7]);
         wind_set(pict_hndl, WF_CXYWH, gl_rmsg[4],
              gl_rmsg[5], gl_rmsg[6], gl_rmsg[7]);
         }
         break;
    case WM_MOVED:
         curr_hndl = wind_find(mousex,mousey);
         if(curr_hndl == pict_hndl)
         {
         gl_rmsg[4] = align_x(gl_rmsg[4]);
         wind_set(pict_hndl, WF_CXYWH, align_x(gl_rmsg[4]) - 1,
         gl_rmsg[5], gl_rmsg[6], gl_rmsg[7]);
         }
         break;
    } /* switch */
    return(done);
} /* hndl_msg */

/*------------------------------*/
/*       do_redraw             */
/*------------------------------*/

VOID
do_redraw(wh, area)            /* redraw specified area from undo bfr */
WORD    wh;
GRECT   *area;
{
    GRECT   box;
    GRECT   dirty_source, dirty_dest;

    graf_mouse(M_OFF, 0x0L);

    wind_get(wh, WF_FIRSTXYWH, &box.g_x, &box.g_y, &box.g_w, &box.g_h);
    while ( box.g_w && box.g_h )
    {
         if (rc_intersect(area, &box))
         {

              if ( wh == pict_hndl )
              {
                   rc_copy(&box, &dirty_dest);
```

```
          if (rc_intersect(&pict_work, &dirty_dest))
          {
            dirty_source.g_x = (dirty_dest.g_x - pict_work.g_x)
                    + pict_undo.g_x;
            dirty_source.g_y = (dirty_dest.g_y - pict_work.g_y)
                    + pict_undo.g_y;
            dirty_source.g_w = dirty_dest.g_w;
            dirty_source.g_h = dirty_dest.g_h;
            rast_op(3, &dirty_source, &pict_mfdb,
            &dirty_dest, &scrn_mfdb);
          }
     }


     if (wh == data_hndl)
     {
          rc_copy(&box, &dirty_dest);
          if (rc_intersect(&data_work, &dirty_dest))
          {
            dirty_source.g_x = (dirty_dest.g_x - data_work.g_x)
                    + data_undo.g_x;
            dirty_source.g_y = (dirty_dest.g_y - data_work.g_y)
                    + data_undo.g_y;
            dirty_source.g_w = dirty_dest.g_w;
            dirty_source.g_h = dirty_dest.g_h;
            rast_op(3, &dirty_source, &data_mfdb,
            &dirty_dest, &scrn_mfdb);
          }
     }


        }
      wind_get(wh, WF_NEXTXYWH, &box.g_x, &box.g_y, &box.g_w, &box.g_h);
    }
    graf_mouse(M_ON, 0x0L);
}

/*------------------------------*/
/*      do_full                 */
/*------------------------------*/
VOID
do_full(wh)   /* depending on current window state, either make window*/
WORD   wh;    /* full size -or- return to previous shrunken size      */
{
GRECT   prev;
GRECT   curr;
GRECT   full;

graf_mouse(M_OFF,0x0L);
wind_get(wh, WF_CXYWH, &curr.g_x, &curr.g_y, &curr.g_w, &curr.g_h);
wind_get(wh, WF_PXYWH, &prev.g_x, &prev.g_y, &prev.g_w, &prev.g_h);
```

```
wind_get(wh, WF_FXYWH, &full.g_x, &full.g_y, &full.g_w, &full.g_h);

if ( rc_equal(&curr,&full) )   /* is full now so shrink it */

   {
    graf_shrinkbox(prev.g_x, prev.g_y, prev.g_w, prev.g_h,
                   full.g_x, full.g_y, full.g_w, full.g_h);
    wind_set(wh, WF_CXYWH, prev.g_x, prev.g_y, prev.g_w, prev.g_h);

    if ( wh == pict_hndl )
       {
       rc_copy(&pict_prev_undo,&pict_undo);
       set_pict_work(TRUE);
       }
    if ( wh == data_hndl)
       {
       rc_copy(&data_prev_undo,&data_undo);
       set_data_work();
       }

    if ((prev.g_x == full.g_x) && (prev.g_y == full.g_y))

    if ( wh == pict_hndl )  do_redraw(wh, &pict_work);
    if ( wh == data_hndl )  do_redraw(wh, &data_work);

   }

else                             /* is not full so make  it full  */

    {
    if (curr_hndl == pict_hndl) rc_copy(&pict_undo,&pict_prev_undo);
    if (curr_hndl == data_hndl) rc_copy(&data_undo,&data_prev_undo);

    graf_growbox(curr.g_x, curr.g_y, curr.g_w, curr.g_h,
                 full.g_x, full.g_y, full.g_w, full.g_h);
    wind_set(wh, WF_CXYWH, full.g_x, full.g_y, full.g_w, full.g_h);
    if (curr_hndl == pict_hndl)  set_pict_work(TRUE);
    if (curr_hndl == data_hndl)  set_data_work(TRUE);

    }
graf_mouse(M_ON,0x0L);
}
```

```
/****************************************************************************/
/****************************************************************************/
/****                                                                    ****/
/****                 Mouse Handling                                     ****/
/****                                                                    ****/
/****************************************************************************/
/****************************************************************************/


/*------------------------------*/
/*      hndl_mouse              */
/*------------------------------*/
WORD
hndl_mouse()
{
    BOOLEAN   done;

    if (m_out)
         graf_mouse(ARROW, 0x0L);
    else
         graf_mouse(monumber, mofaddr);

    m_out = !m_out;
    done = FALSE;
    return(done);
}


/****************************************************************************/
/****************************************************************************/
/****                                                                    ****/
/****                 Button Handling                                    ****/
/****                                                                    ****/
/****************************************************************************/
/****************************************************************************/



/*------------------------------*/
/*      draw_pencil             */
/*------------------------------*/
WORD
draw_pencil(x, y)
UWORD   x, y;
{
    UWORD    pxy[4];
    WORD     done;
    UWORD    mflags;
    UWORD    locount, hicount;
    UWORD    ev_which, bbutton, kstate, kreturn, breturn;
```

```
set_clip(TRUE, &curr_work);
pxy[0] = x;
pxy[1] = y;

vsl_color(vdi_handle,SURE_shade);
vswr_mode(vdi_handle,MD_REPLACE);
vsl_type (vdi_handle,FIS_SOLID);

if (SURE_shade != PEN_ERASER)
{
     vsl_width (vdi_handle,SURE_pen-2);
     vsl_ends(vdi_handle, 2, 2);
     hicount = 0;
     locount = 125;
     mflags = MU_BUTTON | MU_M1 | MU_TIMER;
     graf_mouse(M_OFF, 0x0L);
}
else
{
     vsf_interior(vdi_handle, 1);
     vsf_color(vdi_handle, WHITE);
     mflags = MU_BUTTON | MU_M1;
}

done = FALSE;
while (!done)
{
     ev_which = evnt_multi(mflags,
     0x01, 0x01, 0x00,
     1, pxy[0], pxy[1], 1, 1,
     0, 0, 0, 0, 0,
     ad_rmsg, locount, hicount,
     &pxy[2], &pxy[3], &bbutton, &kstate,
     &kreturn, &breturn);

     if (ev_which & MU_BUTTON)
     {
          if (!(mflags & MU_TIMER))
               graf_mouse(M_OFF, 0x0L);
          if (SURE_shade != PEN_ERASER)
/*             v_pline(vdi_handle, 2, (WORD *) pxy);  */
               v_arc(pict_hndl,x,y,radius,0,3599);
          else
               eraser((WORD) pxy[2], (WORD) pxy[3]);
          graf_mouse(M_ON, 0x0L);
          done = TRUE;
     }
     else
```

```
            if (ev_which & MU_TIMER)
            {
                graf_mouse(M_ON, 0x0L);
                mflags = MU_BUTTON | MU_M1;
            }
            else
            {
                if (!(mflags & MU_TIMER))
                    graf_mouse(M_OFF, 0x0L);
                if (SURE_shade != PEN_ERASER)
                {
/*                  v_pline(vdi_handle, 2, (WORD *) pxy); */
                v_arc(pict_hndl,x,y,radius,0,3599);
                    mflags = MU_BUTTON | MU_M1 | MU_TIMER;
                }
                else
                {
                    eraser((WORD) pxy[2], (WORD) pxy[3]);
                    graf_mouse(M_ON,0x0L);
                }
                pxy[0] = pxy[2];
                pxy[1] = pxy[3];
            }
    } /* while */

    set_clip(FALSE, &curr_work);
    if ( curr_hndl == pict_hndl )  save_pict_work();
    if ( curr_hndl == data_hndl )  save_data_work();
}



/*------------------------------*/
/*      eraser                  */
/*------------------------------*/
VOID
eraser(x, y)                    /* erase rectangle of eraser size at x,y */
WORD    x, y;
{
    WORD    erase_xy[4];

    if (SURE_pen == PEN_FINE)
    {
        erase_xy[0] = x - 2;
        erase_xy[1] = y - 1;
        erase_xy[2] = x + 2;
        erase_xy[3] = y + 1;
     }
```

```
    else
        if (SURE_pen == PEN_MEDIUM)
        {
             erase_xy[0] = x - 4;
             erase_xy[1] = y - 2;
             erase_xy[2] = x + 4;
             erase_xy[3] = y + 2;
        }
        else
        {
             erase_xy[0] = x - 6;
             erase_xy[1] = y - 3;
             erase_xy[2] = x + 6;
             erase_xy[3] = y + 3;
        }
    vr_recfl(vdi_handle, erase_xy);
}


/*************************************************************************/
/*************************************************************************/
/****                                                                ****/
/****          SURE Event Handler                                    ****/
/****                                                                ****/
/*************************************************************************/
/*************************************************************************/

/*------------------------------*/
/*      SURE                    */
/*------------------------------*/
SURE()
{
    BOOLEAN   done;

    key_input = FALSE;
    done = FALSE;
    FOREVER
    {
         ev_which = evnt_multi( MU_MESAG | MU_M1,/* MU_KEYBD*/
         0x02, 0x01, 0x01,
         m_out,
         (UWORD) data_work.g_x, (UWORD) data_work.g_y,
         (UWORD) data_work.g_w, (UWORD) data_work.g_h,
         0, 0, 0, 0, 0,
         ad_rmsg, 0, 0,
         &mousex, &mousey, &bstate, &kstate,
         &kreturn, &bclicks);

         wind_update(BEG_UPDATE);
```

```
        if (!(ev_which & MU_KEYBD))
        {
            if (key_input)
            {
                curs_off();
                key_input = FALSE;
                save_data_work();
            }
        }

        if (ev_which & MU_MESAG)
        if (hndl_msg())
            break;


        if (ev_which & MU_M1)
        if (hndl_mouse())
            break;

        if (ev_which & MU_KEYBD)
        if (hndl_keyboard())
            break;

        wind_update(END_UPDATE);
    }
}



/*************************************************************************/
/*************************************************************************/
/****                                                                ****/
/****           Termination                                          ****/
/****                                                                ****/
/*************************************************************************/
/*************************************************************************/


/*------------------------------*/
/*   SURE_term                  */
/*------------------------------*/
SURE_term(term_type)
WORD    term_type;
{
    switch (term_type)              /* NOTE: all cases fall through */
    {
        case (0 /* normal termination */):
            do_close(pict_hndl, gl_wfull/2, gl_hfull/2);
            wind_delete(pict_hndl);

            do_close(data_hndl, gl_wfull/2, gl_hfull/2);
```

```
          wind_delete(data_hndl);

     case (3):
          menu_bar(0x0L, FALSE);
          dos_free(pict_mfdb.mp);
     case (2):
          v_clsvwk( vdi_handle );
     case (1):
          wind_update(END_UPDATE);
          appl_exit();
     case (4):
          break;
   }
}


/************************************************************************/
/************************************************************************/
/****                                                               ****/
/****              Initialization                                   ****/
/****                                                               ****/
/************************************************************************/
/************************************************************************/
/*------------------------------*/
/*       pict_init              */
/*------------------------------*/
VOID
pict_init()
{
   LONG     tree;
   WORD     tr_obj, nobj;

   rsrc_gaddr(R_TREE, SUREINFO, &tree);
   trans_gimage(tree, SUREIMG);
   rsrc_gaddr(R_TREE, SUREPEND, &tree);
   for (tr_obj = SUREPFIN; tr_obj <= SUREEBRD; tr_obj++)
   {
        trans_gimage(tree, tr_obj);
        LWSET(OB_TYPE(tr_obj), G_USERDEF);
        nobj = tr_obj - SUREPFIN;
        brushub[nobj].ub_code = drawaddr;
        brushub[nobj].ub_parm = LLGET(OB_SPEC(tr_obj));
        LLSET(OB_SPEC(tr_obj), ADDR(&brushub[nobj]));
   }
}
```

```
/*-------------------------------*/
/*      SURE_init              */
/*-------------------------------*/
WORD
SURE_init()
{
    WORD    work_in[11];
    WORD    i,pict_w,data_w,data_x;
    GRECT   box;
    LONG    tree;

    gl_apid = appl_init();             /* initialize libraries        */
    if (gl_apid == -1)
         return(4);
    wind_update(BEG_UPDATE);
    graf_mouse(HOUR_GLASS, 0x0L);
    if (!rsrc_load( ADDR("SURE.RSC") ))
    {
        graf_mouse(ARROW, 0x0L);
        form_alert(1,
        ADDR("[3][Fatal Error !|SURE.RSC|File Not Found][ Abort ]"));
        return(1);
     }

                                       /* open virtual workstation    */
    for (i=0; i<10; i++)  work_in[i]=1;
                          work_in[10]=2;

    gem_handle = graf_handle(&gl_wchar,&gl_hchar,&gl_wbox,&gl_hbox);
    vdi_handle = gem_handle;
    v_opnvwk(work_in,&vdi_handle,work_out);
    if (vdi_handle == 0)          return(1);

    scrn_width = work_out[0] + 1;  /* WIDTH  OF SCREEN  in pixels   */
    scrn_height = work_out[1] + 1; /* HEIGHT OF SCREEN  in pixels   */
    scrn_xsize = work_out[3];
    scrn_ysize = work_out[4];

    char_fine = work_out[46];
    char_medium = work_out[48];
    char_broad = char_medium * 2;

    vq_extnd(vdi_handle, 1, work_out);
    scrn_planes = work_out[4];

    scrn_area.g_x = 0;
    scrn_area.g_y = 0;
    scrn_area.g_w = scrn_width;
    scrn_area.g_h = scrn_height;
    scrn_mfdb.mp = 0x0L;
```

```
/* menu bar operations */

    pict_init();                /* transforms & user defined objects  */
    ad_rmsg = ADDR((BYTE *) &gl_rmsg[0]);
    rsrc_gaddr(R_TREE, SUREMENU, &gl_menu);  /* init menu */
    menu_bar(gl_menu, TRUE);                 /* show menu */



    wind_get(DESK,WF_WXYWH,&gl_xfull,&gl_yfull,&gl_wfull,&gl_hfull);
                                /* workarea of DESK window     */


                             /* define buffer area for picture window */
    pict_mfdb.fwp = 1*( scrn_width);
    pict_mfdb.fww = pict_mfdb.fwp>>4;
    pict_mfdb.fh  = 1*( scrn_height);
    pict_mfdb.np  = scrn_planes;
    pict_mfdb.ff  = 0;

    pict_rect.g_x = 0;
    pict_rect.g_y = 0;
    pict_rect.g_w = pict_mfdb.fwp;
    pict_rect.g_h = pict_mfdb.fh;

    size_pict_mfdb = (LONG)((pict_mfdb.fwp>>3)+1) * (LONG)pict_mfdb.fh *
                     (LONG)pict_mfdb.np;
    loc_pict_mfdb =
                  pict_mfdb.mp  = dos_alloc(size_pict_mfdb);
    if (pict_mfdb.mp == 0)   return(2);


                                /* define buffer area for data window */
    data_mfdb.fwp = scrn_width;
    data_mfdb.fww = data_mfdb.fwp>>4;
    data_mfdb.fh  = 1*( scrn_height);
    data_mfdb.np  = scrn_planes;
    data_mfdb.ff  = 0;

    data_rect.g_x = 0;
    data_rect.g_y = 0;
    data_rect.g_w = data_mfdb.fwp;
    data_rect.g_h = data_mfdb.fh;


    size_data_mfdb = (LONG)((data_mfdb.fwp>>3)+1) * (LONG)data_mfdb.fh *
                (LONG)data_mfdb.np;
    loc_data_mfdb =
                  data_mfdb.mp  = dos_alloc(size_data_mfdb);
```

```
if (data_mfdb.mp == 0)
    return(2);

                /* set all pixels in pict buffer to zero (white) */
rast_op(0,&pict_rect,&pict_mfdb,&pict_rect,&pict_mfdb);


                /* set all pixels in data buffer to zero (white) */
rast_op(0,&data_rect,&data_mfdb,&data_rect,&data_mfdb);


              /* create the two windows 'picture' and 'data' */

pict_hndl=wind_create(0x0fef,gl_xfull-1,gl_yfull,gl_wfull,gl_hfull);
data_hndl=wind_create(0x0fef,gl_xfull-1,gl_yfull,gl_wfull,gl_hfull);

if  ( (pict_hndl == -1) || (data_hndl == -1) )
    {
    form_alert(1, string_addr(SURENWDW));
    return(3);
    }

 graf_mouse(HOUR_GLASS, 0x0L);

 wind_set(pict_hndl, WF_NAME,
 (WORD) LLOWD(ADDR(wdw_title)),(WORD) LHIWD(ADDR(wdw_title)), 0, 0);
 wind_set(data_hndl, WF_NAME,
 (WORD) LLOWD(ADDR(wdw_tdata)),(WORD) LHIWD(ADDR(wdw_tdata)), 0, 0);

 pict_w = (6*gl_wfull)/10    ;  /* init width of pict window */
 data_w = (4*gl_wfull)/10 - 2;  /* init width of data window */
 data_x = pict_w + 2;           /* init x-coord of data window */

 do_open(data_hndl,gl_wfull/2,gl_hfull/2,data_x,gl_yfull,
 data_w, gl_hfull);
 do_open(pict_hndl,gl_wfull/2,gl_hfull/2,gl_xfull,gl_yfull,
 pict_w, gl_hfull);

wind_get(pict_hndl,WF_WXYWH,&pict_work.g_x,&pict_work.g_y,
                            &pict_work.g_w,&pict_work.g_h);
                             /*  init workarea of pict window     */
wind_get(data_hndl,WF_WXYWH,&data_work.g_x,&data_work.g_y,
                            &data_work.g_w,&data_work.g_h);
                              /* init workarea of data window     */


pict_undo.g_x = 0;                          /* init pict_undo area */
pict_undo.g_y = 0;
pict_undo.g_w = pict_work.g_w;
pict_undo.g_h = pict_work.g_h;
```

```
    data_undo.g_x = 0;                          /* init data_undo area */
    data_undo.g_y = 0;
    data_undo.g_w = data_work.g_w;
    data_undo.g_h = data_work.g_h;

    rc_copy(&pict_undo,&pict_prev_undo);  /* init. 'previous' undo    */
    rc_copy(&data_undo,&data_prev_undo);  /* areas for full/unfulling */


    set_data_work(TRUE);          /* init slider bars of data window  */
    rst_data_work();              /* data window turns white          */
    init_data();                  /* init data structures             */

    set_pict_work(TRUE);           /* init slider bars of pict window */
    rst_pict_work();              /* pict window turns white          */



     graf_mouse(ARROW,0x0L);
     wind_update(END_UPDATE);

     objc_xywh(gl_menu,SUREDESK,&box);
     rsrc_gaddr(R_TREE,INTRO,&tree);
     hndl_dial(tree,0,box.g_x,box.g_y,box.g_w,box.g_y);
     desel_obj(tree,INTROOK);

     return(0);
}



/************************************************************************/
/************************************************************************/
/****                                                               ****/
/****            Main Program                                       ****/
/****                                                               ****/
/************************************************************************/
/************************************************************************/


/*------------------------------*/
/*      GEMAIN                  */
/*------------------------------*/
GEMAIN()
{
    WORD    term_type;

    if (!(term_type = SURE_init()))
        SURE();
    SURE_term(term_type);
}
```

APPENDIX C

SURE OPTIONS LISTING

This APPENDIX contains all the code required to enter and label the Semi-markov models displayed in the Data and Picture windows. The code is all original and is self documenting.

```
/*************************************************************************/
/*                                                                       */
/*      This is the SURE-options portion of the SURE program. All       */
/*      routines necessary to implement the following functions are    */
/*      contained in the routine;                                        */
/*              Enter Constants                                          */
/*              Enter States                                             */
/*              Enter Transitions                                        */
/*              Reliplot   - Set-up to be done at later date             */
/*              Hardcopy   - Set-up to be done at later date             */
/*                                                                       */
/*      John c. Bordeaux                                                 */
/*      Naval Postgraduate School                                        */
/*      Master's Thesis                                                  */
/*                                                                       */
/*************************************************************************/


/*------------------------------*/
/*   includes                   */
/*------------------------------*/

#include "portab.h"                   /* portable coding conv        */
#include "machine.h"                  /* machine depndnt conv        */
#include "obdefs.h"                   /* object definitions          */
#include "treeaddr.h"                 /* tree address macros         */
#include "gembind.h"                  /* gem binding structs         */
#include "sure.h"                     /* SURE  apl  resource         */
#include "math.h"                     /* lattice c                   */
#include "limits.h"                   /* lattice c                   */
#include "optstrct.h"

#define  TE_TXTLEN(x) (x+24)
/*------------------------------*/
/*    preprocessor utilities    */
/*------------------------------*/
#define    MALLOC(x)     ((x *) malloc(sizeof(x)))
#define    CALLOC(n, x)  ((x *) calloc(n, sizeof(x)))

/*------------------------------*/
/*      OPT/SURE defines        */
/*------------------------------*/
#define    END_UPDATE   0
#define    BEG_UPDATE   1
#define    MAXSTATE     100          /* maximum number of states      */
#define    MAXTRANS     200          /* maximum number of transition */
#define    ETCALC       0            /* Whites algebraic formula,E(t)*/
#define    START        1            /* indicates the start state    */
#define    GEOMETRIC    FALSE        /* deselect geometric           */
#define    prn:         EPSHSS10.FNT
```

```
/*------------------------------*/
/*   External Functions         */
/*------------------------------*/

EXTERN    LONG    dos_alloc();
EXTERN    UWORD   DOS_ERR;

/*------------------------------*/
/*    Global Data Structures    */
/*------------------------------*/

extern    struct state_loc state_array[MAXSTATE];

/*------------------------------*/
/*   external variables         */
/*------------------------------*/

extern    WORD    gl_wchar;          /* character width              */
extern    WORD    gl_hchar;          /* character height             */
extern    WORD    gl_wbox;           /* box (cell) width             */
extern    WORD    gl_hbox;           /* box (cell) height            */
extern    WORD    gl_rmsg[8];        /* message buffer               */
extern    WORD    gl_hspace;         /* Ht. of space between lines   */
extern    WORD    SURE_pen;
extern    WORD    SURE_shade;
extern    WORD    PEN_ERASE;
extern    WORD    TIME;              /* mission time                 */
extern    WORD    POINTS;            /* number of pts plotted/calc   */
extern    LONG    PRUNE;             /* deactivate the pruning       */
extern    WORD    WARNDIG;           /* set UB accuracy              */
extern    WORD    TRUNC;             /* set loop traversals at 3     */
extern    WORD    LBFACT;            /* sets the Ki and Kj = 20      */
extern    WORD    LIST;              /* sets the output level        */
extern    WORD    title;
extern    WORD    ydata;             /* Current data line            */
extern    WORD    radius;            /* Cureent radius of state      */
extern    WORD    gem_handle;        /* GEM vdi handle               */
extern    WORD    vdi_handle;        /* SURE vdi handle              */
extern    WORD    work_out[57];      /* open virt workstation values*/
extern    WORD    ev_which;          /* event multi return state(s) */
extern    WORD    vdi_handle;
extern    WORD    curr_hndl;
extern    WORD    pict_hndl;         /* SURE window handle           */
extern    WORD    data_hndl;         /* SURE data window handle      */
extern    WORD    pen_shade ;        /* saved pen shade              */
extern    WORD    SURE_height ;      /* SURE current char height     */
extern    WORD    char_fine;         /* character height for fine    */
extern    WORD    char_medium;       /* character height for medium */
extern    WORD    char_broad;        /* character height for broad   */
extern    WORD    monumber ;         /* mouse form number            */
extern    LONG    mofaddr ;          /* mouse form address           */
```

```
extern    WORD    file_handle;        /* file handle -> pict ld/sv   */
extern    WORD    key_xbeg;           /* x posit for line beginning  */
extern    WORD    key_ybeg;           /* y posit for line beginning  */
extern    WORD    key_xcurr;          /* current x position          */
extern    WORD    key_ycurr;          /* current y position          */
extern    WORD    runno;              /* run number                  */
extern    WORD    pathcount;          /* number of paths             */
extern    WORD    cnttrunc;           /* number of loops truncated   */
extern    WORD    cntprune;           /* number of paths pruned      */
extern    WORD    bigst;              /* largest state entered so far*/
extern    WORD    i;
extern    WORD    num_states;         /* number of states entered    */

extern    UWORD   m_out ;             /* mouse in/out of window flag */
extern    UWORD   mousex, mousey;     /* mouse x,y position          */
extern    UWORD   bstate, bclicks;    /* button state, & # of clicks */
extern    UWORD   kstate, kreturn;    /* key state and keyboard char */

extern    LONG    ad_rmsg;            /* LONG pointer to message bfr */
extern    LONG    gl_menu;            /* menu tree address           */
extern    LONG    drawaddr;

extern    GRECT   curr_work;          /* work area of current window */
extern    GRECT   pict_work;          /* work area of picture window */
extern    GRECT   pict_rect;          /* work area of picture window */
extern    GRECT   data_work;          /* work area of data window    */
extern    GRECT   curr_undo;
extern    GRECT   pict_undo;

extern    double  speclow;            /* variable range value        */
extern    double  spechigh;           /* variable range value        */
extern    double  specval;            /* special variable value      */
extern    double  ubfail;             /* prob of system failure      */
extern    double  lbfail;             /* lower bound of failure      */
extern    double  e_of_t;             /* E(T) computation result     */
extern    double  e_of_t_del;         /* E(T-*) computation result   */
extern    double  pialpha;            /* product of the alphas       */
extern    double  lowbf,lowbg;        /* intermediate results        */

extern    BOOLEAN    key_input;       /* key inputting state         */
extern    BOOLEAN    erun;            /* runtime errors flag         */
extern    BOOLEAN    nonlinear;       /* nonlinear flag              */
extern    BOOLEAN    et_bad;          /* E(T) comp. is inaccurate    */
extern    BOOLEAN    rec_slow;        /* recovery too slow           */
extern    BOOLEAN    std_big;         /* Rec Std. too big            */
extern    BOOLEAN    rate_big;        /* Exp. rate too fast          */
```

```
/*************************************************************************/
/*                  DATA INPUT SECTION                                   */
/*************************************************************************/

/*------------------------------*/
/*       init_data              */
/*------------------------------*/
init_data()
{
   WORD    i;


   for ( i = 1; i <= MAXSTATE; i++)
   {
        state_array[i].x_coord =  0;
        state_array[i].y_coord =  0;
   }
}/* init_data */

/*------------------------------*/
/*       do_constants           */
/*------------------------------*/
VOID
do_constants()                             /* This routine defines the    */
{                                          /* values of the constants     */
    GRECT   box;
    LONG    tree;
    LONG    cons_addr,valu_addr;
    BYTE    dname[13],dvalu[25];
    WORD    exit_obj;

   save_data_work();
   set_clip(FALSE,&pict_work);
   set_data_work(TRUE);
   set_clip(TRUE,&data_work);

   FOREVER
   {
   objc_xywh(gl_menu,SUREOPTS,&box);
   rsrc_gaddr(R_TREE,ENTERCON,&tree);
   exit_obj = hndl_dial(tree,0,box.g_x, box.g_y, box.g_w, box.g_h);
   desel_obj(tree,CONSTYES);

   if ( exit_obj == CONSTNO)
        {
        desel_obj(tree,CONSTNO);
        rst_data_work();
        set_pict_work(TRUE);
        set_clip(TRUE,&pict_work);
        return;
        }
```

```
    objc_xywh(gl_menu,SUREOPTS,&box);
    rsrc_gaddr(R_TREE,ADDCONST,&tree);

    cons_addr = LLGET(OB_SPEC(CONSNAME));
    valu_addr = LLGET(OB_SPEC(CONSVALU));

    LLSET( cons_addr, ADDR(dname) );
    LLSET( valu_addr, ADDR(dvalu) );

    LWSET( TE_TXTLEN(cons_addr), 9);
    LWSET( TE_TXTLEN(valu_addr),21);

    dname[0] = '\0';
    dvalu[0] = '\0';
    if(hndl_dial(tree,CONSNAME,box.g_x, box.g_y, box.g_w, box.g_h)
             == ADDCONOK )

    rst_data_work();

    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
             &gl_wchar , gl_hchar);
    v_gtext(data_hndl, data_work.g_x+5,data_work.g_y+ydata*8,dname);
    v_gtext(data_hndl, data_work.g_x+80,data_work.g_y+ydata*8,"=");
    v_gtext(data_hndl, data_work.g_x+90,data_work.g_y+ydata*8,dvalu);
    vst_height(data_hndl ,char_medium, &gl_wchar, &gl_hchar,
             &gl_wchar , gl_hchar);

    save_data_work();
    rst_data_work();
    desel_obj(tree,ADDCONOK);

    ydata++;
    }
}


/*************************************************************************/
/*                                                                       */
/*      This is the Enter states portion of the Sure program.            */
/*                                                                       */
/*************************************************************************/

/*------------------------------*/
/*      do_states               */
/*------------------------------*/
VOID
do_states()                          /* This is the Work Horse of    */
{                                    /* enter states.                */
    GRECT   sta_area ,box;
    LONG    tree;
```

```
    graf_mouse(monumber,mofaddr);

    sta_area.g_x = pict_work.g_x + 35; /* This box defines an area on */
    sta_area.g_y = pict_work.g_y + 19; /* the screen to insure states */
    sta_area.g_w = pict_work.g_w - 50; /* do not overlap the boundav  */
    sta_area.g_h = pict_work.g_h - 30;

    while (num_states <= MAXSTATE)
    {

        ev_which = evnt_multi(MU_BUTTON | MU_M1 ,
        0x01, 0x01, 0x01,
        1,
        (UWORD) pict_work.g_x, (UWORD) pict_work.g_y,
        (UWORD) pict_work.g_w, (UWORD) pict_work.g_h,
        0, 0, 0, 0, 0,
        ad_rmsg,0,0,
        &mousex, &mousey, &bstate, &kstate,
        &kreturn, &bclicks);



        if ( ev_which & MU_BUTTON)
        {
            if ((inside(mousex,mousey,&sta_area)) &&
                (!too_close(mousex,mousey,num_states)))
                 {
                 draw_state(mousex,mousey,num_states);
                 evnt_timer(500,0);

                 state_array[num_states].x_coord = mousex;
                 state_array[num_states].y_coord = mousey;

  /*             state_array[num_states].x_coord = pict_undo.g_x +
                            ( mousex - pict_work.g_x);
                 state_array[num_states].y_coord = pict_undo.g_y +
                            ( mousey - pict_work.g_y);
  */             num_states++ ;
                 }

        }
        if ( ev_which & MU_M1)
           {
           save_pict_work();
           return;
           }
    }
}
```

```
/*-------------------------------*/
/*      draw_state               */
/*-------------------------------*/
WORD
draw_state(x, y ,num_s)                /* This routine draws and      */
WORD    num_s;                         /* labels the state at the     */
UWORD   x, y;                          /* x, y, location. The state   */
{                                      /* is labeled with the value   */
   UWORD   x_next,y_next;              /* contained in num_s.         */
   UWORD   xt,yt;
   BYTE    num_str[20];

   graf_mouse(M_OFF, 0x0L);            /* Turn mouse off              */
   itoa(num_s,num_str);                /* Convert num_s to a string   */


/*  radius = 16;  */

   if (num_s <= 9)                     /* The next set of statements  */
       {                               /* determine where to label    */
       xt = x - 3;                     /* the state for medium.       */
       yt = y + 2;
       }
   if ((num_s >= 10) && (num_s <= 99))
       {
       xt = x - 8;
       yt = y + 2;
       }
   if ( num_s >= 100)
       {
       xt = x - 12;
       yt = y + 2;
       }

       vsl_width(pict_hndl,1);
       vsl_ends(pict_hndl,2,2);
       v_arc(pict_hndl,x,y,radius,0,3599);       /* Draw state here   */
/*     draw_pencil(x,y);    */
       vsl_width(pict_hndl,1);
       v_gtext(pict_hndl, xt, yt, num_str);      /* Label state here  */

   graf_mouse(M_ON, 0x0L);             /* Turn mouse on               */
}

/*-------------------------------*/
/*      guess                    */
/*-------------------------------*/
guess(x_curr,y_curr,x_next,y_next,ptg) /* guess the next state locat. */
UWORD  x_curr,y_curr,x_next,y_next;
GRECT  *ptg;
{
```

```
    if(( x_curr + 60) <= (ptg->g_x + ptg->g_w))
         {
         x_next = x_curr + 60;
         y_next = y_curr;
         }
    else
         x_next = x_curr;
         y_next = y_curr + 60;

}/*  guess */



/*------------------------------*/
/*       too_close              */
/*------------------------------*/
too_close(x ,y ,num )
UWORD    x, y;                          /* This routine determines if  */
WORD     num;                           /* the states are too close,   */
{                                       /* it takes the absolute value */
    WORD   i;                           /* of the difference between   */
    UWORD  x_temp, y_temp;              /* the present mouse position  */
                                        /* and state positions that    */
                                        /* area stored in an array     */
    if ( num == 1 )                     /* called state_array.         */
        return(FALSE);

    num = num - 1;
    for ( i = 1; i <= num; i++)
    {

    x_temp = abs( x - state_array[i].x_coord);
    y_temp = abs( y - state_array[i].y_coord);

    if(( x_temp <=  75) && (y_temp <= 40))
         return(TRUE);
    }
         return(FALSE);

} /* too_close*/
```

```
/*********************************************************************/
/*                                                                   */
/*   This is the transitions section of Sure. The routines used to   */
/*   draw and input the data for the transitions between states is   */
/*   included in this section.                                       */
/*                                                                   */
/*********************************************************************/

/*------------------------------*/
/*      get_slow()              */
/*------------------------------*/
BYTE
*get_slow(froms,tos)                  /* dialogue box input rate     */
WORD   froms,tos;
{
    LONG    tree;
    LONG    ted_addr;
    GRECT   box;
    BYTE    *name;
    double  atof();
    VALREC  mean,stdev,fract;
    save_data_work();

    objc_xywh(gl_menu,SUREOPTS,&box);
    rsrc_gaddr(R_TREE,SURESLOW,&tree);
    ted_addr = LLGET(OB_SPEC(SLOWLAMB));
    LLSET( ted_addr, ADDR(name) );
    LWSET( TE_TXTLEN(ted_addr), 9);

    if (hndl_dial(tree,SLOWLAMB,box.g_x,box.g_y,box.g_w,box.g_h)
                 == SLOWOK)
         {
              desel_obj(tree,SLOWOK);
              mean.base  = atof(name);
              mean.coef  = atof(name);
              stdev.base = -1.0;
              stdev.coef = -1.0;
              fract.base = 1.0;
              fract.coef = 1.0;
              enter(froms,tos,&mean,&stdev,&fract);
              return(name);
         }
} /* get_slow() */
```

```
/*-------------------------------*/
/*      get_fast()               */
/*-------------------------------*/
BYTE
get_fast(froms,tos,mean,stdev,frac)    /* dialogue box input rate     */
    WORD    froms,tos;
    BYTE    *mean,*stdev,*frac;
{
    LONG    tree;
    LONG    mean_addr;
    LONG    stdev_addr;
    LONG    frac_addr;
    GRECT   box;
    double  atof();
    VALREC  tmean,tstdev,tfract;
    save_data_work();

    objc_xywh(gl_menu,SUREOPTS,&box);
    rsrc_gaddr(R_TREE,SUREFAST,&tree);

    mean[0]  = '\0';
    mean_addr = LLGET(OB_SPEC(SUREMEAN));
    LLSET( mean_addr, ADDR(mean) );
    LWSET( TE_TXTLEN(mean_addr), 9);

    stdev[0] = '\0';
    stdev_addr = LLGET(OB_SPEC(SURESTD));
    LLSET( stdev_addr, ADDR(stdev) );
    LWSET( TE_TXTLEN(stdev_addr), 9);

    frac[0]  = '\0';
    frac_addr = LLGET(OB_SPEC(SUREFRAC));
    LLSET( frac_addr, ADDR(frac) );
    LWSET( TE_TXTLEN(frac_addr), 5);

    if (hndl_dial(tree,SUREMEAN,box.g_x,box.g_y,box.g_w,box.g_h)
                == FASTOK)
         {
              desel_obj(tree,FASTOK);
              tmean.base  = atof(mean);
              tmean.coef  = atof(mean);
              tstdev.base = atof(stdev);
              tstdev.coef = atof(stdev);
              tfract.base = atof(frac);
              tfract.base = atof(frac);
              enter(froms,tos,&tmean,&tstdev,&tfract);
         }
} /* get_fast() */
```

```
/*------------------------------*/
/*      do_fast_transition      */
/*------------------------------*/
VOID                                    /* This routine determines the */
do_fast_transition()                    /* froms and the tos  for input*/
{                                       /* for input of transition data*/
    WORD    exit_obj,froms ,tos;
    GRECT   box;
    LONG    pxy[4],tree;                /* define pxy array as a long  */
    double  dblpxy[4];                  /* redefine pxy array as double*/
    BYTE    mean[13],stdev[13],frac[9];

    set_pict_work(TRUE);
    set_clip(TRUE, &pict_work);
    graf_mouse(monumber,mofaddr);

         ev_which = evnt_multi(MU_BUTTON | MU_M1 ,
                   0x01, 0x01, 0x01,
                   1,
                   (UWORD) pict_work.g_x, (UWORD) pict_work.g_y,
                   (UWORD) pict_work.g_w, (UWORD) pict_work.g_h,
                   0, 0, 0, 0, 0,
                   ad_rmsg,0,0,
                   &mousex, &mousey, &bstate, &kstate,
                   &kreturn, &bclicks);

         if ( ev_which & MU_M1)
            {                           /* exit if mouse out of window */
            save_pict_work();
            return;
            }

         if ( ev_which & MU_BUTTON)
         {
             if  (which_state(mousex,mousey,&froms))
                  {
                  pxy[0] = state_array[froms].x_coord ;
                  pxy[1] = state_array[froms].y_coord ;
                  dblpxy[0] = pxy[0];      /* store x(froms)-location*/
                  dblpxy[1] = pxy[1];      /* store y(froms)-location*/
                  evnt_timer(500,0);
                  }
                           /* After from state is selected loop    */
             do            /* until a good to state is selected    */
                  {
                   ev_which = evnt_multi(MU_BUTTON | MU_M1 ,
                       0x01, 0x01, 0x01,
                       1,
                       (UWORD) pict_work.g_x, (UWORD) pict_work.g_y,
                       (UWORD) pict_work.g_w, (UWORD) pict_work.g_h,
                       0, 0, 0, 0, 0,
```

```
                ad_rmsg,0,0,
                &mousex, &mousey, &bstate, &kstate,
                &kreturn, &bclicks);

               if ( ev_which & MU_M1)
                    {         /* exit if mouse out of window */
                     save_pict_work();
                     return;
                    }
           } while (!(( ev_which & MU_BUTTON) &&
              ( which_state(mousex,mousey,&tos))));

           pxy[2] = state_array[tos].x_coord ;
           pxy[3] = state_array[tos].y_coord ;
           dblpxy[2] = pxy[2];
           dblpxy[3] = pxy[3];

           draw_tran(&dblpxy,froms,tos);
           save_pict_work();
           save_data_work();

                      /* Prepare to get the transition data    */
                      /* draw dialog box for fast transition   */

           get_fast(froms,tos,&mean,&stdev,&frac);

                                  /* label the data window      */
           lab_fast_window(froms,tos,&mean,&stdev,&frac);
                                  /* label the fast transition  */
           label_fast(&pxy,froms,tos,&mean,&stdev,&frac);
                rst_pict_work();
                rst_data_work();
                set_pict_work(TRUE);
                set_clip(TRUE, &pict_work);
                return;
       }
         return;

} /* do_fast_transition */



/*------------------------------*/
/*       do_slow_transition     */
/*------------------------------*/
VOID                                    /* This routine determines the */
do_slow_transition()                    /* froms and the tos  for      */
{                                       /* input of transition data    */
    WORD    i ,froms ,tos;
    GRECT   box;
    LONG    pxy[4],tree;                /* define pxy array as a long  */
```

```
double  dblpxy[4];              /* redefine pxy array as double*/
BYTE    cname[13],*get_slow();
BYTE    pstr[13];

graf_mouse(monumber,mofaddr);

set_clip(TRUE, &pict_work);



    ev_which = evnt_multi(MU_BUTTON | MU_M1 ,
               0x01, 0x01, 0x01,
               1,
               (UWORD) pict_work.g_x, (UWORD) pict_work.g_y,
               (UWORD) pict_work.g_w, (UWORD) pict_work.g_h,
               0, 0, 0, 0, 0,
               ad_rmsg,0,0,
               &mousex, &mousey, &bstate, &kstate,
               &kreturn, &bclicks);


    if ( ev_which & MU_M1)
        {                       /* exit if mouse out of window */
        save_pict_work();
        return;
        }
    if ( ev_which & MU_BUTTON)
    {
        if  (which_state(mousex,mousey,&froms))
            {
            pxy[0] = state_array[froms].x_coord ;
            pxy[1] = state_array[froms].y_coord ;
            dblpxy[0] = pxy[0];      /* store x(froms)-location*/
            dblpxy[1] = pxy[1];      /* store y(froms)-location*/
            evnt_timer(500,0);
            }
                    /* After from state is selected loop    */
        do          /* until a good to state is selected    */
            {
             ev_which = evnt_multi(MU_BUTTON | MU_M1 ,
                 0x01, 0x01, 0x01,
                 1,
                 (UWORD) pict_work.g_x, (UWORD) pict_work.g_y,
                 (UWORD) pict_work.g_w, (UWORD) pict_work.g_h,
                 0, 0, 0, 0, 0,
                 ad_rmsg,0,0,
                 &mousex, &mousey, &bstate, &kstate,
                 &kreturn, &bclicks);

                 if ( ev_which & MU_M1)
```

```
                        {       /* exit if mouse out of window */
                        save_pict_work();
                        return;
                        }
                } while (!(( ev_which & MU_BUTTON) &&
                   ( which_state(mousex,mousey,&tos))));

                pxy[2] = state_array[tos].x_coord ;
                pxy[3] = state_array[tos].y_coord ;
                dblpxy[2] = pxy[2];
                dblpxy[3] = pxy[3];

                draw_tran(&dblpxy,froms,tos); /* draw arrow         */

                        /* Prepare to get the transition data   */
                        /* draw dialog box for slow transition  */
                save_pict_work();
                save_data_work();
                strcpy(cname , get_slow(froms,tos));
                rst_data_work();
                rst_pict_work();

                save_pict_work();
                lab_slow_window(froms,tos,&cname);
                save_data_work();

                label_slow(&pxy,froms,tos,&cname);
                                /* Label slow transition        */
        }
          set_clip(FALSE,&pict_work);
          return;
} /* do_slow_transition */


/*------------------------------*/
/*      which_state             */
/*------------------------------*/
which_state(x ,y ,curr_st)
UWORD   x, y;                   /* This routine determines the */
WORD    *curr_st;               /* state which is under the    */
{                               /* mouse cursor                */
    WORD   i,num;
    UWORD  x_temp, y_temp;


    num = num_states - 1;
    for ( i = 1; i <= num; i++)
    {
    x_temp = abs( x - state_array[i].x_coord);
    y_temp = abs( y - state_array[i].y_coord);
```

```
    if(( x_temp <=  18) && (y_temp <= 18))
         {
         *curr_st = i;
         return(TRUE);                   /* return TRUE if state is part*/
         }                               /* state_array                 */
    }
    return(FALSE);
} /* which_state */


/*------------------------------*/
/*        draw_tran(pxy)        */
/*------------------------------*/
VOID
draw_tran(dblpxy,froms,tos)              /* This routine draws the line */
double   *dblpxy;                        /* between the states.         */
WORD     froms,tos;
{
    double y_diff, x_diff;               /* Distance between froms and  */
    double  angle ;                      /* tos.                        */
    LONG   lngpxy[4];
    WORD   pxy[4];
    double sin(),cos(),atan2();

    y_diff = dblpxy[3] - dblpxy[1];      /* Compute distance, y1 - y2   */
    x_diff = dblpxy[2] - dblpxy[0];      /* Compute distance, x1 - x2   */

    if ( x_diff != 0 )
        angle = atan2(y_diff , x_diff);/* Compute angle                */
    else
        angle = 1.570796327;
                                         /* find the edge of the circle */
    lngpxy[0] = cos(angle) * 16.0 + dblpxy[0];
    lngpxy[1] = sin(angle) * 10.0 + dblpxy[1];
    lngpxy[2] = dblpxy[2] - cos(angle) * 16.0;
    lngpxy[3] = dblpxy[3] - sin(angle) * 10.0;
       pxy[0] = lngpxy[0];               /* convert to integer          */
       pxy[1] = lngpxy[1];
       pxy[2] = lngpxy[2];
       pxy[3] = lngpxy[3];

/*  vswr_mode(pict_hndl,MD_REPLACE);
*/  vsl_width(pict_hndl,1);
    vsl_color(vdi_handle,SURE_shade);
    vsl_ends(pict_hndl,2,1);
    v_pline(pict_hndl, 2,(WORD*)pxy); /* draw arrow between states    */
    vsl_width(pict_hndl,1);
    vsl_ends(pict_hndl,2,2);
} /* draw_tran */
```

```
/*------------------------------*/
/*      label_fast_transition  */
/*------------------------------*/
VOID
label_fast(pxy,froms,tos,mean,stdev,frac)
                                       /* This routine labels a fast  */
LONG     *pxy;                         /* transition between states.  */
WORD     froms,tos;
BYTE     *mean,*stdev,*frac;

{
    WORD   y_diff, x_diff;             /* Distance between froms and  */
    WORD   xm,ym,xs,ys,xf,yf;          /* tos.                        */

    set_pict_work(TRUE);
    set_clip(FALSE,&data_work);

    y_diff = pxy[3] - pxy[1];          /* Compute distance, y1-> y2   */
    x_diff = pxy[2] - pxy[0];          /* Compute distance, x1-> x2   */

    xm = .35 * x_diff + pxy[0];
    ym = .35 * y_diff + pxy[1];

    xs = .5 * x_diff + pxy[0];
    ys = .5 * y_diff + pxy[1];

    xf = .65 * x_diff + pxy[0];
    yf = .65 * y_diff + pxy[1];

    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
    v_gtext(data_nndl,xm+3,ym,mean);
    v_gtext(data_hndl,xs+3,ys,stdev);
    v_gtext(data_hndl,xf+3,yf,frac);
    vst_height(data_hndl ,char_medium , &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);

    save_pict_work();
    rst_pict_work();
    set_pict_work(TRUE);
    set_clip(TRUE,&pict_work);

} /* label_fast */
```

```
/*-------------------------------*/
/*       label_slow_transition   */
/*-------------------------------*/
VOID
label_slow(pxy,froms,tos,cname)         /* This routine labels a slow  */
LONG     *pxy;                          /* transition between states.  */
WORD     froms,tos;
BYTE     *cname;
{

    WORD   x1,y1;


    x1 = pxy[0] + 25;
    y1 = pxy[1] - 5;

    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
    v_gtext(data_hndl,x1,y1,cname);
    vst_height(data_hndl ,char_medium, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);

    save_pict_work();
    rst_pict_work();
} /* label_slow */



/*--------------------------------*/
/*       lab_fast_window()        */
/*--------------------------------*/
VOID
lab_fast_window(froms,tos,mean,stdev,frac)
                                        /* This routine writes the      */
                                        /* information that has been    */
WORD    froms,tos;                      /* entered to the data window   */
BYTE    *mean,*stdev,*frac;

{
    BYTE    froms_str[5],tos_str[5];

    set_data_work(TRUE);
    set_clip(FALSE,&pict_work);
    rst_data_work();

    itoa(froms,froms_str);              /* Convert froms to a string    */
    itoa(tos,tos_str);                  /* Convert tos to a string      */

    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
    v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,froms_str);
```

```
    v_gtext(data_hndl, data_work.g_x+25,data_work.g_y+ydata*8,"To");
    v_gtext(data_hndl, data_work.g_x+50,data_work.g_y+ydata*8,tos_str);
    v_gtext(data_hndl, data_work.g_x+75,data_work.g_y+ydata*8,mean);
    v_gtext(data_hndl,data_work.g_x+130,data_work.g_y+ydata*8,stdev);
    v_gtext(data_hndl,data_work.g_x+190,data_work.g_y+ydata*8,frac);
    vst_height(data_hndl ,char_medium, &gl_wchar, &gl_hchar,
            &gl_wchar , gl_hchar);

    save_data_work();
    rst_data_work();

    ydata++;

    set_pict_work(TRUE);
    set_clip(TRUE,&pict_work);
    rst_pict_work();
} /* lab_fast_window() */



/*--------------------------------*/
/*       lab_slow_window()        */
/*--------------------------------*/
VOID
lab_slow_window(froms,tos,cname)        /* This routine writes the     */
                                        /* information that has been   */
WORD    froms,tos;                      /* entered to the data window  */
BYTE    *cname;
{
    BYTE    froms_str[5],tos_str[5];

    set_data_work(TRUE);
    set_clip(TRUE,&data_work);
    rst_data_work();

    itoa(froms,froms_str);              /* Convert froms to a string   */
    itoa(tos,tos_str);                  /* Convert tos to a string     */

    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
            &gl_wchar , gl_hchar);
    v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,froms_str);
    v_gtext(data_hndl,data_work.g_x+25,data_work.g_y+ydata*8,"To");
    v_gtext(data_hndl,data_work.g_x+50,data_work.g_y+ydata*8,tos_str);
    v_gtext(data_hndl,data_work.g_x+75,data_work.g_y+ydata*8,cname);
    vst_height(data_hndl ,char_medium, &gl_wchar, &gl_hchar,
            &gl_wchar , gl_hchar);

    save_data_work();
    rst_data_work();
```

```
    ydata++;

    set_pict_work(TRUE);
    set_clip(TRUE,&pict_work);
    rst_pict_work();
} /* lab_slow_window() */



/*------------------------------*/
/*      do_plot                 */
/*------------------------------*/
VOID
do_plot()
{
    save_data_work();
    set_clip(FALSE,&pict_work);
    set_data_work(TRUE);
    set_clip(TRUE,&data_work);
    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
    ydata++;
    v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
           "PLOT RELIABILITY HAS BEEN CALLED");  ydata++;
    vst_height(data_hndl ,char_medium, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
    save_data_work();
    rst_data_work();
    set_pict_work(TRUE);
    set_clip(TRUE,&pict_work);
}



/*------------------------------*/
/*      do_hardcopy             */
/*------------------------------*/
VOID
do_hardcopy()
{
    save_data_work();
    set_clip(FALSE,&pict_work);
    set_data_work(TRUE);
    set_clip(TRUE,&data_work);
    vst_height(data_hndl ,char_fine, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
    ydata++;
    v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
           "DO HARD_COPY HAS BEEN CALLED");  ydata++;
    vst_height(data_hndl ,char_medium, &gl_wchar, &gl_hchar,
               &gl_wchar , gl_hchar);
```

```
    save_data_work();
    rst_data_work();
    set_pict_work(TRUE);
    set_clip(TRUE,&pict_work);
}
```

APPENDIX D

SURE COMPUTATION LISTING

This Appendix contains the modules required to do the SURE Computations. The modules are generally the original SURE computations modules converted from Pascal to C. Several additional modules were required for the interface into the GEM environment.

The enclosed listings are not the final listings of the SURE program. The latest listings are available on disk.

The computation module contains debugging aids, the module is not fully functional.

```
/*************************************************************************/
/*                                                                       */
/*      This is the SURE-Compute portion of the SURE program. All       */
/*      routines necessary to implement the following functions are     */
/*      contained in the routine;                                       */
/*                Computation                                           */
/*                                                                       */
/*      John c. Bordeaux                                                */
/*      Naval Postgraduate School                                       */
/*      Master's Thesis                                                 */
/*                                                                       */
/*************************************************************************/


/*------------------------------*/
/*  includes                    */
/*------------------------------*/
/*#include <stdio.h>  */
#include "portab.h"                     /* portable coding conv         */
#include "machine.h"                    /* machine depndnt conv         */
#include "obdefs.h"                     /* object definitions           */
#include "treeaddr.h"                   /* tree address macros          */
#include "gembind.h"                    /* gem binding structs          */
#include "sure.h"                       /* SURE  apl  resource          */
#include "math.h"                       /* lattice c                    */
#include "limits.h"                     /* lattice c                    */
#include "optstrct.h"

#define  TE_TXTLEN(x) (x+24)
/*------------------------------*/
/*   preprocessor utilities     */
/*------------------------------*/
#define    MALLOC(x)    ((x *) alloc(sizeof(x)))
#define    CALLOC(n, x) ((x *) calloc(n, sizeof(x)))

/*------------------------------*/
/*      OPT/SURE defines        */
/*------------------------------*/
#define    END_UPDATE   0
#define    BEG_UPDATE   1
#define    MAXSTATE     100            /* maximum number of states      */
#define    MAXTRANS     200            /* maximum number of transition */
#define    ETCALC       0              /* Whites algebraic formula,E(t)*/
#define    START        1              /* indicates the start state     */
#define    GEOMETRIC    FALSE          /* deselect geometric            */
#define    prn:         EPSHSS10.FNT
#define    minreal      0.29e-38
#define    maxreal      1.70e38
#define    idleng       8
#define    NULLA        0               /* pointer value for error      */
#define    ALLOCSIZE    1000            /* size of available space      */
```

```
static char allocbuf[ALLOCSIZE];      /* storage for alloc          */
static char *allocp = allocbuf;       /* next free position         */

/*------------------------------*/
/*    External Functions        */
/*------------------------------*/

EXTERN    LONG    dos_alloc();
EXTERN    UWORD   DOS_ERR;

/*------------------------------*/
/*     Global Data Structures   */
/*------------------------------*/

BOOLEAN   predst[MAXSTATE];
extern    struct state_loc state_array[MAXSTATE];
DEATHREC  *fdeath;
TRANREC   *succst[MAXSTATE];
INCON     *inconlist;
WORD      TIME = 10;                  /* mission time               */
WORD      POINTS = 25;                /* number of pts plotted/calc */
LONG      PRUNE = 0.0;                /* deactivate the pruning     */
WORD      WARNDIG = 1;                /* set UB acc. to two digits  */
WORD      TRUNC  = 3;                 /* set loop traversals at 3   */
WORD      LBFACT = 20;                /* sets the Ki and Kj = 20    */
WORD      LIST = 2;                   /* sets the ouput level       */

/*------------------------------*/
/*    external variables        */
/*------------------------------*/

extern    WORD    gl_wchar;           /* character width            */
extern    WORD    gl_hchar;           /* character height           */
extern    WORD    gl_wbox;            /* box (cell) width           */
extern    WORD    gl_hbox;            /* box (cell) height          */
extern    WORD    gl_rmsg[8];         /* message buffer             */
extern    WORD    gl_hspace;          /* Ht. of space between lines */
extern    WORD    SURE_pen;
extern    WORD    SURE_shade;
extern    WORD    PEN_ERASE;
extern    WORD    title;
extern    WORD    ydata;              /* Current data line          */
extern    WORD    radius;             /* Cureent radius of state    */
extern    WORD    gem_handle;         /* GEM vdi handle             */
extern    WORD    vdi_handle;         /* SURE vdi handle            */
extern    WORD    work_out[57];       /* open virt workstation values*/
extern    WORD    ev_which;           /* event multi return state(s) */
extern    WORD    vdi_handle;
extern    WORD    curr_hndl;
extern    WORD    pict_hndl;          /* SURE window handle         */
extern    WORD    data_hndl;          /* SURE data window handle    */
```

```
extern   WORD    pen_shade ;          /* saved pen shade              */
extern   WORD    SURE_height ;        /* SURE current char height     */
extern   WORD    char_fine;           /* character height for fine    */
extern   WORD    char_medium;         /* character height for medium  */
extern   WORD    char_broad;          /* character height for broad   */
extern   WORD    monumber ;           /* mouse form number            */
extern   LONG    mofaddr ;            /* mouse form address           */
extern   WORD    file_handle;         /* file handle -> pict ld/sv    */
extern   WORD    key_xbeg;            /* x posit for line beginning   */
extern   WORD    key_ybeg;            /* y posit for line beginning   */
extern   WORD    key_xcurr;           /* current x position           */
extern   WORD    key_ycurr;           /* current y position           */
extern   WORD    runno;               /* run number                   */
extern   WORD    pathcount;           /* number of paths              */
extern   WORD    cnttrunc;            /* number of loops truncated    */
extern   WORD    cntprune;            /* number of paths pruned       */
extern   WORD    bigst;               /* largest state entered so far*/
extern   WORD    i;
extern   WORD    num_states;          /* number of states entered     */

extern   UWORD   m_out ;              /* mouse in/out of window flag  */
extern   UWORD   mousex, mousey;      /* mouse x,y position           */
extern   UWORD   bstate, bclicks;     /* button state, & # of clicks  */
extern   UWORD   kstate, kreturn;     /* key state and keyboard char  */

extern   LONG    ad_rmsg;             /* LONG pointer to message bfr  */
extern   LONG    gl_menu;             /* menu tree address            */
extern   LONG    drawaddr;

extern   GRECT   curr_work;           /* work area of current window  */
extern   GRECT   pict_work;           /* work area of picture window  */
extern   GRECT   pict_rect;           /* work area of picture window  */
extern   GRECT   data_work;           /* work area of data window     */
extern   GRECT   curr_undo;
extern   GRECT   pict_undo;

extern   double  speclow;             /* variable range value         */
extern   double  spechigh;            /* variable range value         */
extern   double  specval;             /* special variable value       */
extern   double  ubfail;              /* prob of system failure       */
extern   double  lbfail;              /* lower bound of failure       */
extern   double  e_of_t;              /* E(T) computation result      */
extern   double  e_of_t_del;          /* E(T-*) computation result    */
extern   double  pialpha;             /* product of the alphas        */
extern   double  lowbf,lowbg;         /* intermediate results         */

extern   BOOLEAN    key_input;        /* key inputting state          */
extern   BOOLEAN    erun;             /* runtime errors flag          */
extern   BOOLEAN    nonlinear;        /* nonlinear flag               */
extern   BOOLEAN    et_bad;           /* E(T) comp. is inaccurate     */
```

```
extern    BOOLEAN   rec_slow;      /* recovery too slow        */
extern    BOOLEAN   std_big;       /* Rec Std. too big         */
extern    BOOLEAN   rate_big;      /* Exp. rate too fast       */



/*--------------------------------*/
/*      init_pointers()           */
/*--------------------------------*/
init_pointers()                    /* initiate succst array    */
{
    WORD    i;
    BYTE    num_str[20];
    fdeath = 0L;
ltoa((LONG)fdeath,num_str);
dos_printf(num_str);

    for( i = 0; i <= MAXSTATE; i++)
         succst[i] = NULLPTR;
} /* init_pointers() */


/*--------------------------------*/
/*      alloc()                   */
/*--------------------------------*/
BYTE
*alloc(n)                     /* return pointer to n characters   */
WORD   n;
{
    if( allocp + n <= allocbuf + ALLOCSIZE )     /* fits           */
    {
         allocp += n;
         return(allocp - n);          /* old p                    */
    }
    else                              /* not enough room          */
         return(NULLA);
} /* alloc() */


/*--------------------------------*/
/*      free()                    */
/*--------------------------------*/
free(p)                            /* free storage pointed to by p*/
BYTE   *p;
{
    if(p >= allocbuf && p < allocbuf + ALLOCSIZE )
         allocp = p;
} /* free() */
```

```
/*-------------------------------*/
/*       power()                 */
/*-------------------------------*/
WORD
power(x,n)                              /* Raise x to n-th power; n>0  */
WORD    x, n;
{
    WORD    i, p;
    p = 1;
    for ( i=1; i <= n; ++i )
         p = p * x;
    return(p);
} /* power() */




/*-------------------------------*/
/*       printds()               */
/*-------------------------------*/
VOID
printds(n,string)
WORD     n;
BYTE     string[64];
{
    WORD i, ii;
    BYTE tempst[64];

    i = 0;
    ii = 0;

    if( n < 0 )
    {
         n = -n;
         string[ii++] = '-';
    }

    do
    {
         tempst[i++] = n % 10 + '0';
    } while( (n /= 10) > 0 );

    while( --i >= 0 )
         string[ii++] = tempst[i];
    string[ii] = '\0';
} /* printds() */
```

```
/*-------------------------------*/
/*      fota()                   */
/*-------------------------------*/
VOID
fota( n, s)                          /* Convert n to characters in s*/
BYTE     s[64];
double   n;
{
    WORD   i,ii, sign;
    WORD   m, pow;
    BYTE   string[64];
    double x, tempx;

    i = 0;
    if( n < 0 )                      /* record sign                 */
    {
        s[i++] = '-';
        n = -n;                      /* make n positive             */
    }

    pow = 0;
    while( n < 1. )
    {
        pow--;
        n = n*10;
    }
    while( n > 10.0 )
    {
        pow++;
        n = n/10;
    }
    m = (int) n;
    printds(m,string);
    strcpy(s+i,string);
    i = strlen(s);
    s[i++] = '.';

    x = n-m;
    while( (x != 0.0 ) && (i < 8) )
    {
        tempx = x;
        tempx *= 10;
        ii = (int) tempx;
        s[i++] = ii + 48;
        x = tempx - ii;
    }
```

```
    if( pow != 0 )
    {
        s[i++] = 'E';
        if( pow >= 0 )
            s[i++] = '+';
        printds(pow,string);
        strcpy(s+i,string);
    }
    else s[i] = '\0';
    return;
} /* fota() */



/*------------------------------*/
/*       findtran               */
/*------------------------------*/
BOOLEAN
findtran(froms,tos,pmean,pstdev,pfract)
statetypes  froms, tos;
VALREC      *pmean, *pstdev, *pfract;
{
    TRANREC   *p;
    BOOLEAN   found;
dos_printf("Findtran has been entered");
    {
    p =  succst[froms];
    found = FALSE;
    while ( p != NULLPTR)
    {
        if( p->st == tos)             /* Transition found           */
        {
            found = TRUE;
            pmean->coef  = p->mean.coef;
            pmean->base  = p->mean.base;
            pstdev->coef = p->stdev.coef;
            pstdev->base = p->stdev.base;
            pfract->coef = p->fract.coef;
            pfract->base = p->fract.base;
        }
        p = p->next;
    }
    }
dos_printf("Findtran has been exited");
    return(found);
} /* findtran */
```

```
/*-------------------------------*/
/*      itranschk                */
/*-------------------------------*/

WORD
itranschk(froms,tos,nslow,nfast)
WORD   froms,tos;
WORD   nslow,nfast;
{

    TRANREC    *p;
    WORD       found;
dos_printf("Itranschk has been entered");
    {
    p = succst[froms];
    found = 0;
    nslow = 0;
    nfast = 0;
    while (p != NULLPTR)
         {
         if ( p->st = tos)              /* Transition found             */
              found = found + 1;
         if ( p->st >= 0)
              {
              if ( p->stdev.base == -1.0)
                   nslow = nslow + 1;
              else
                   nfast = nfast + 1;
              }
         p = p->next;
         }
    }
dos_printf("Itranschk has been exited");
    return(found);
} /* itranschk */


/*-------------------------------*/
/*      enter()                  */
/*-------------------------------*/
VOID
enter(froms,tos,pmean,pstdev,pfract)
statetypes  froms,tos;
VALREC      *pmean, *pstdev, *pfract;
{
    TRANREC   *p;
    VALREC    v1, v2, v3;
    BOOLEAN   fnd;
    BYTE      num_str[64];
```

```
dos_printf("Enter has been entered");
    set_clip(TRUE,&data_work);
    vst_height(data_hndl,char_fine,&gl_wchar,&gl_hchar,
              &gl_wchar,gl_hchar);
    {

    fnd = findtran(froms,tos,&v1,&v2,&v3);
    if( fnd && (tos == -1) )
    {
         v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*6,
            "**** HOLDING TIME ALREADY ENTERED");
         ydata++;
    }
    else
         {
         predst[tos] = TRUE;
         if( froms > bigst )
              bigst = froms;
         if( tos > bigst )
              bigst = tos;
         if( ( froms > MAXSTATE) || (tos > MAXSTATE)
            || (froms < 0) || (tos < -1) )
         {
              v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*6,
              "**** STATE OUT OF RANGE");
              ydata++;
         }
         else
              {
              p = MALLOC(TRANREC);
              p->st   = tos;
              p->next = NULLPTR;
              p->mean.base  = pmean->base;
              p->mean.coef  = pmean->coef;
              p->stdev.base = pstdev->base;
              p->stdev.coef = pstdev->coef;
              p->fract.base = pfract->base;
              p->fract.coef = pfract->coef;
              p->next = succst[froms];
              succst[froms] = p;
dos_printf("The pointer for p follows");
ltoa((LONG)p,num_str);
dos_printf(num_str);
fota(pmean->base,num_str);
dos_printf(num_str);
fota(pstdev->base,num_str);
dos_printf(num_str);
fota(pfract->base,num_str);
dos_printf(num_str);
              }
         }
```

```
    }
dos_printf("Enter has been exited");
    set_clip(FALSE,&data_work);
    vst_height(data_hndl,char_medium,&gl_wchar,&gl_hchar,
               &gl_wchar,gl_hchar);
} /* enter() */



/*--------------------------------*/
/*       cmp()                    */
/*--------------------------------*/
double
cmp(pv)
VALREC   *pv;
{
dos_printf("cmp has been entered and exited");
    return( pv->base + (pv->coef)*specval );
} /* cmp() */



/*--------------------------------*/
/*       clear                    */
/*--------------------------------*/
VOID
clear()
{
    TRANREC   *t, *l;
    WORD      i;
    {
         for ( i = 1; i <= bigst; i++ )
         {
              if( succst[i] != NULLPTR )
              {
                   t = succst[i];
                   while( t != NULLPTR )
                   {
                        l = t;
                        t = t->next;
                        free(l);
                   }
                   succst[i] = NULLPTR;
              }
              predst[i] = FALSE;
         }
         bigst = -1;
     }
} /* clear */
```

```
/*------------------------------*/
/*      approx_et()             */
/*------------------------------*/
double
approx_et(ep,mt)
EREC     *ep;
double   mt;
{
    double    e_of_t;
    WORD      k;
BYTE    num_str[64];
dos_printf("approx_et has been entered");
        e_of_t = 1;
        k = 0;
        e_of_t_del = 1;
        while( ep != NULLPTR)
        {
dos_printf("The value for k follows");
itoa(k,num_str);
dos_printf(num_str);
            k++;
ltoa((LONG)ep,num_str);
dos_printf(num_str);
dos_printf("The value for ep->lambda follows");
fota(ep->lambda,num_str);
dos_printf(num_str);
dos_printf("The value for e_of_t follows");
fota(e_of_t,num_str);
dos_printf(num_str);
dos_printf("The value for mt follows");
fota(mt,num_str);
dos_printf(num_str);
            e_of_t = ep->lambda * e_of_t * mt/k;
            ep = ep->next;
dos_printf("The pointer for ep,EREC,in the while loop follows");
ltoa((LONG)ep,num_str);
dos_printf(num_str);
        }
fota(e_of_t,num_str);
dos_printf(num_str);
dos_printf("approx_et has been exited");
        return(e_of_t);
} /* approx_et */
```

```
/*-------------------------------*/
/*      returncnt()              */
/*-------------------------------*/
WORD
returncnt(cs,h)
statetypes  cs;
HIST        *h;
{
    WORD    c;
dos_printf("returnct has been entered");
    c = 0;
    while( h != NULLPTR )
    {
        if ( h->st == cs )
        {
            c++;
            h = h->next;
        }
        return( c );
    }
dos_printf("returnct has been exited");
} /* returncnt */



/*-------------------------------*/
/*      hcopy                    */
/*-------------------------------*/
VOID
hcopy(hp,np)
HIST   *hp, *np;
{
    HIST   *op, *lp;
    BYTE   num_str[10];
dos_printf("hcopy has been entered");

    np = NULLPTR;
    while( hp != NULLPTR )
    {
        op = lp;
        lp = MALLOC(HIST);
dos_printf("The pointer for lp,HIST follows");
ltoa((LONG)lp,num_str);
dos_printf(num_str);
        lp->st = hp->st;
        lp->next = NULLPTR;
        if( np == NULLPTR)
            np = lp;
        else
            op->next = lp;
        hp = hp->next;
```

```
    }
dos_printf("hcopy has been exited");
} /* hcopy */



/*-------------------------------*/
/*       ecopy()                  */
/*-------------------------------*/
VOID
ecopy(ep,np)
EREC   *ep, *np;
{
    EREC   *op, *lp;
    BYTE   num_str[10];
dos_printf("ecopy has been entered");

    np = NULLPTR;
    while( ep != NULLPTR )
    {
         op = lp;
         lp = MALLOC(EREC);
dos_printf("The pointer for lp,EREC follows");
ltoa((LONG)lp,num_str);
dos_printf(num_str);
         lp->next = NULLPTR;
         lp->lambda = ep->lambda;
         lp->gamma  = ep->gamma;
         if( np == NULLPTR )
              np = lp;
         else
              op->next = lp;
         ep = ep->next;
    }
dos_printf("ecopy has been exited");
} /* ecopy() */



/*-------------------------------*/
/*       hadd()                   */
/*-------------------------------*/
VOID
hadd(cs,h)
statetypes    cs;
HIST     *h;
{
    HIST    *lp;
    BYTE    num_str[10];
dos_printf("hadd has been entered");
```

```
    lp = h;
    h = MALLOC(HIST);
dos_printf("The pointer for h,EREC follows");
ltoa((LONG)h,num_str);
dos_printf(num_str);
    h->st = cs;
    h->next = lp;
dos_printf("hadd has been exited");
} /* hadd() */



/*--------------------------------*/
/*        addtoelist()            */
/*--------------------------------*/
VOID
addtoelist(cs,cp,ep,nep)
statetypes  cs;
TRANREC     *cp;
EREC        *ep, *nep;
{
    double    gamma;
    TRANREC   *p;
    double    cmp();
    BYTE      num_str[10];
dos_printf("addtoelist has been entered");

    gamma = 0.0;
    p = succst[cs];
    while( p != NULLPTR )                 /* All are exponential          */
    {
        if( p != cp )
            gamma = gamma + cmp(&(p->mean));
        if( cmp(&(p->mean)) > (TIME/10.0) )
            rate_big = TRUE;
        p = p->next;
    }
    nep = MALLOC(EREC);
dos_printf("The pointer for nep,EREC follows");
ltoa((LONG)nep,num_str);
dos_printf(num_str);
    nep->lambda = cmp(&(cp->mean));
    nep->gamma  = gamma;
    nep->next   = ep;
dos_printf("addtoelist has been exited");
} /* addtoelist */
```

```
/*-------------------------------*/
/*      death()                  */
/*-------------------------------*/
BOOLEAN
death(p)
TRANREC  *p;
{
dos_printf("death has been entered");
    if( p == NULLPTR )
    {
         dos_printf("return TRUE");
         return(TRUE);
    }
    else
         {
         dos_printf("return p->st = -1");
         return(( p->st == -1 ) && ( p->next == NULLPTR) );
         }
dos_printf("death has been exited");
} /* death() */



/*-------------------------------*/
/*      addtodeath()             */
/*-------------------------------*/
VOID
addtodeath(ds, ep, hp, lb, ub, delta)
statetypes   ds;
EREC     *ep;
HIST     *hp;
double   lb, ub, delta;
{
    DEATHREC   *d;
    BOOLEAN    found;
    WORD       lc;
    BYTE       lbstr[32],ubstr[32],num_str[15];
dos_printf("addtodeath has been entered");


    if( LIST >= 5 )
    {
         v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                "**** ADDTODEATH ENTERED");
/*       calc_et(ep,delta);  */
         ub = ub*e_of_t;
         lb = lb*e_of_t_del;
         ydata++;
    }
```

```
    if( LIST >= 2 )
    {
         itoa(d->st,num_str);
         v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 num_str);
         fota(lb,lbstr);
         v_gtext(data_hndl,data_work.g_x+83,data_work.g_y+ydata*8,
                 lbstr);
         fota(ub,ubstr);
         v_gtext(data_hndl,data_work.g_x+161,data_work.g_y+ydata*8,
                 ubstr);
         ydata++;

         if( et_bad )
         {
              v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                      "**** E(T) INACCURATE");
              ydata++;
         }
         et_bad = FALSE;
    }
    d = fdeath;
    found = FALSE;
    while( d != NULLPTR )
    {
         if( d->st == ds )
         {
              found = TRUE;
              d->lprob = d->lprob + lb;
              d->uprob = d->uprob + ub;
         }
         d = d->next;
    }
    if( !found )
    {
         d = MALLOC(DEATHREC);
dos_printf("The pointer for d,DEATHREC follows");
ltoa((LONG)d,num_str);
dos_printf(num_str);
         d->next  = NULLPTR;
         d->st    = ds;
         d->lprob = lb;
         d->uprob = ub;
         if( fdeath == NULLPTR )
              fdeath = d;
         else
         {
              d->next = fdeath;
              fdeath  = d;
         }
    }
```

```
    if( LIST >= 3 )
    {
         v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 "**** ADDTO DEATH ENTERED");
              ydata++;
         lc = 1;
         while( hp != NULLPTR )
         {
              v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                      "----> WRITE OUT THE STATE");
              ydata++;
              if( (lc % 10) == 0 )
              {
         v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 "**** PRINT OUT THE ERROR");
              ydata++;
              }
              hp = hp->next;
              lc++;
         }
    }
    save_data_work();
dos_printf("addtodeath has been exited");
} /* addtodeath */



/*-------------------------------*/
/*       allexpo()               */
/*-------------------------------*/
BOOLEAN
allexpo(lp,uc,sig2c)
TRANREC  *lp;
double   uc, sig2c;
{
    BOOLEAN  tallexpo;
    double   cmp();

dos_printf("allexpo has been entered");

    tallexpo = TRUE;
    uc = -1;
    sig2c = -1;
    while( lp != NULLPTR )
    {
         if( lp->st >= 0 )
         {
              if( cmp(&(lp->stdev)) >= 0 )
                   tallexpo = FALSE;
         }
```

```
          else
          {
               uc    = cmp(&(lp->mean));
               sig2c = cmp(&(lp->stdev));
               sig2c = sig2c * sig2c;
          }
          lp = lp->next;
     }
dos_printf("allexpo has been exited");
     return(tallexpo);
} /* allexpo */



/*--------------------------------*/
/*      allexpolee()              */
/*--------------------------------*/
BOOLEAN
allexpolee(p)
TRANREC   *p;
{
     BOOLEAN   tallexpolee;

     tallexpolee = TRUE;
     while( p != NULLPTR )
     {
          if( p->st >= 0 )
          {
               if( cmp( &(p->fract)) > 0 )
                    tallexpolee = FALSE;
          }
          p = p->next;
     }
     return(tallexpolee);
} /* allexpolee() */



/*--------------------------------*/
/*      traverse()                */
/*--------------------------------*/
VOID
traverse(cs,hp,ep,lb,ub,delta)
statetypes    cs;
HIST     *hp;
EREC     *ep;
double   lb, ub, delta;
{
   TRANREC   *lp, *p;
   EREC      *lep, *nep;
   HIST      *nhp;
```

```
    WORD      retc,returncnt();
    double    alpha, uc, sig2c;
    double    mn, std, frac, tmom, f1, f2;
    double    delta_at_cs, ri, sj;
    double    oldlb, oldub, sumfract, sumexpos, pmax;
    double    approx_et(),cmp();
    BYTE      num_str[20],pmaxstr[32];

dos_printf("Traverse has been entered");
    retc = returncnt(cs,hp);
    if( LIST >= 5 )
        v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
              "--- TRAVERSE --- CS, LB, UB");
              ydata++;
    if( retc>= TRUNC )
    {
        cnttrunc++;
        if( LIST >= 4 )
        {
            pmax = ub*approx_et(ep,TIME);
dos_printf("ep following pmax in traverse follows");
ltoa((LONG)ep,num_str);
dos_printf(num_str);
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    "Loop At State");
            itoa(cs,num_str);
            v_gtext(data_hndl,data_work.g_x+85,data_work.g_y+ydata*8,
                    num_str);
            fota(pmax,pmaxstr);
            v_gtext(data_hndl,data_work.g_x+95,data_work.g_y+ydata*8,
                    pmaxstr);
            ydata++;
            goto lab999;
        }
    }
    else
    {
        pmax = ub*approx_et(ep,TIME);
dos_printf("ep following pmax in traverse follows");
ltoa((LONG)ep,num_str);
dos_printf(num_str);
        if( pmax <= PRUNE )
        {
            cntprune++;
            if( LIST >= 4 )
            {
              v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                      "Path Prune at");
              itoa(cs,num_str);
              v_gtext(data_hndl,data_work.g_x+85,data_work.g_y+ydata*8,
                      num_str);
```

```
              ydata++;
             }
             goto lab999;
         }
     }

     nhp = MALLOC(HIST);
dos_printf("The pointer for nhp,HIST follows");
ltoa((LONG)nhp,num_str);
dos_printf(num_str);
     nhp->st = cs;
     nhp->next = hp;
     p = succst[cs];
dos_printf("The pointer for p,succst[] follows");
ltoa((LONG)p,num_str);
dos_printf(num_str);

     if ( death(p) )
     {
         addtodeath(cs,ep,nhp,lb,ub,delta);
         pathcount++;
     }
     else if( allexpo(p,uc,sig2c) )
     {
         while( p != NULLPTR )
         {
             if( p->st >= 0 )
             {
dos_printf("ep prior to addtoelist in traverse follows");
ltoa((LONG)ep,num_str);
dos_printf(num_str);
dos_printf("nep prior to addtoelist in traverse follows");
ltoa((LONG)nep,num_str);
dos_printf(num_str);
                 addtoelist(cs,p,ep,nep);
dos_printf("nep after call to addtoelist in traverse follows");
ltoa((LONG)nep,num_str);
dos_printf(num_str);
dos_printf("nhp prior to call to traverse in traverse follows");
ltoa((LONG)nhp,num_str);
dos_printf(num_str);
                 traverse(p->st,nhp,nep,lb,ub,delta);
dos_printf("The pointer for nep,EREC, to be freed follows");
ltoa((LONG)nep,num_str);
dos_printf(num_str);
                 free(nep);
             }
         p = p->next;
         }
     }
```

```
else
{         /* -- At least one transition from P not exponential -- */
    oldlb = lb;
    oldub = ub;

    if( sig2c < 0 )        /* Holding Info Not Found By Allexpo     */
    {                      /* Approximate From Transition Info      */
        uc = 0;
        sig2c = 0;
        lp = succst[cs];
        sumfract = 0.0;
        sumexpos = 0.0;
        while( lp != NULLPTR )
        {
            if( lp->st >= 0 )       /* Not Holding Time Info       */
            {
                mn  = cmp(&(lp->mean));
                std = cmp(&(lp->stdev));
                if ( std >= 0 )     /* Fast Recovery Transition    */
                {
                    frac = cmp(&(lp->fract));
                    uc = uc + frac*mn;
                    sumfract = sumfract + frac;
                    tmom = std*std + mn*mn;
                    sig2c = sig2c + frac*tmom;
                }
                else                /* Slow Transition             */
                    sumexpos = sumexpos + mn;
            }
        lp = lp->next;
        }
        if( abs(1.0-sumfract) > 1.0e-10 )       /* Machine Dependent */
        {
            erun = TRUE;
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    "**** ERROR **** SUM of EXITING");
            ydata++;
        }
        sig2c = sig2c - uc*uc;
    }

    if( LIST >= 5 )
    {
        v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                "Write UC, Sig2c, and Sumexpos");
        ydata++;
    }

    delta_at_cs = delta;
    while( p != NULLPTR )
    {
```

```
lb = oldlb;
ub = oldub;
if( p->st >= 0 )
{
    mn = cmp(&(p->mean));
    std = cmp(&(p->stdev));
    tmom = mn*mn + std*std;
    if( std >= 0 )          /* --- Fast On Path ---        */
    {
        frac = cmp(&(p->fract));
        ub = ub*frac;
        if( LBFACT > 0 )
            ri = LBFACT*(mn+std);
        else
            ri = sqrt(mn);
        if( ri <= minreal )
        {
            fl = 0.0;
   v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
           "Instantaneous Recovery Not Allowed");
            ydata++;
        }
        else
            fl = ( 1.0-sumexpos*mn - tmom/(ri*ri) );
        if( fl <= 0 )
        {
            if( LIST >= 5 )
            {
   v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
           "LIST >= 5 in Traverse, SO on AND ON");
                ydata++;
            }
            fl = 0.0;
            if( sumexpos > 0.1 )
                rate_big = TRUE;
            else if( (tmom / (ri*ri)) > 1 )
                std_big = TRUE;
            else
                rec_slow = TRUE;
        }
        lb = lb*frac*fl;
        if( LIST >= 5 )
        {
   v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
           "Ri and Delta");
            ydata++;
        }
        delta = delta_at_cs + ri;
    }
    else
```

```
 {                       /* Slow on path transition     */
     alpha = cmp(&(p->mean));
     ub = ub*alpha*uc;
     if( LBFACT > 0 )
         sj = LBFACT*(uc+sqrt(sig2c));
     else
         sj = sqrt(uc);
     tmom = uc*uc + sig2c;
     if( sj <= minreal )
     {
         f2 = 0.0;
 v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
        "Instantaneous Recovery Not Allowed");
         ydata++;
     }
     else
         f2 = uc - tmom*( sumexpos/2.0 + 1.0/sj );

     if( f2 <= 0 )
     {
         f2 = 0.0;
         if( sumexpos > 0.1 )
             rate_big = TRUE;
         else if( uc > 0.1 )
             rec_slow = TRUE;
         else
             std_big = TRUE;
         if( LIST >= 5 )
         {
 v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
        "**** F2 Negative ****");
         ydata++;
         itoa(cs,num_str);
v_gtext(data_hndl,data_work.g_x+85,data_work.g_y+ydata*8,
         num_str);
         ydata++;

         lp = succst[cs];
         while( lp != NULLPTR )
         {
             if( lp->st >= 0 )     /* Not Holdin Time   */
             {
                 mn  = cmp(&(lp->mean));
                 std = cmp(&(lp->stdev));
 v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
        "**** To State ****");
                 itoa(lp->st,num_str);
v_gtext(data_hndl,data_work.g_x+105,data_work.g_y+ydata*8,
                 num_str);
                 ydata++;
             }
```

```
                        lp = lp->next;
                    }
                    }               /* if listlevel = 5          */
                }
            lb = lb*alpha*f2;
            if( LIST >= 5 )
            {
           v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                   "**** SJ,DELTA ****");
                ydata++;
            }
            delta = delta_at_cs + sj;
            }
        if( delta >= TIME )
        {
         v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                "**** DELTA > TIME ****");
            ydata++;
        }
        traverse(p->st,nhp,ep,lb,ub,delta);
        }
    p = p->next;
    } /* while */
  } /* if */
dos_printf("The pointer for nhp,HIST, to be freed follows");
ltoa((LONG)nhp,num_str);
dos_printf(num_str);
  free(nhp);
  lab999:  ;
  save_data_work();
dos_printf("Traverse has been exited");
} /* traverse() */



/*------------------------------*/
/*      hdr1()                  */
/*------------------------------*/
VOID
hdr1()
{
    INCON   *incp;
    WORD    ic,icnt;


    if( LIST > 0 )
    {
/*      if( leeflag == LEE )
        {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    "----- LEE STATISTICAL ANALYSIS MODE ----");
```

```
            ydata++;
        }     */             /* Lee Analysis Mode To Be Added Later    */
        incp = inconlist;
        icnt = 0;
        while( incp != NULLPTR )
        {
            icnt++;
            for( i = 1; i <= idleng; i++ )
                if( incp->id[i] == ' ' )
                    goto lab7;
                else
                {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                      incp->id[i]);
                ydata++;
                }
lab7:       v_gtext(data_hndl,data_work.g_x+45,data_work.g_y+ydata*8,
                      incp->v);
                ydata++;
           if( (icnt % 3) == 0 )
                {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                      " NULL in HDR1");
                ydata++;
                }
           incp = incp->next;
        }
        if( (icnt % 3) != 0 )
           {
           v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 " If hdr1 used more code required");
           ydata++;
           }
    }
} /* hdr1(0 */



/*-------------------------------*/
/*      hdr2()                   */
/*-------------------------------*/
VOID
hdr2()
{

    v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*6,
            "DEATHSTATE   LOWERBOUND   UPPERBOUND");
    ydata++;
    save_data_work();
} /* hdr2() */
```

```
/*-------------------------------*/
/*      hdr3()                   */
/*-------------------------------*/
VOID
hdr3()
{

    v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
            "HEADER NUMBER THREE CALLED");
    ydata++;
    save_data_work();
} /* hdr3() */



/*-------------------------------*/
/*      getstart()               */
/*-------------------------------*/
WORD
getstart()
{
    WORD   i, pos;

    pos = -1;
    for( i = 0; i <= bigst; i++ )
    {
         if( !predst[i] )
         {
              if( succst[i] != NULLPTR )
              {
                   if( pos >= 0 )
                   {
                        pos = -2;
                        goto lab99;
                   }
                   else
                        pos = i;
              }
         }
    }
lab99:   return(pos);
} /* getstart() */
```

```
/*-------------------------------*/
/*      do_compute()             */
/*-------------------------------*/
VOID
do_compute()
{
DEATHREC     *d;
statetypes   ds, startstate;
BOOLEAN      pwarn;
BYTE         num_str[5],lprobstr[32],uprobstr[32];
BYTE         ubfailstr[32],lbfailstr[32];
double       lb, ub, del;
BYTE         *testptr;
dos_printf("do_compute has been entered");

    set_clip(TRUE,&data_work);
    vst_height(data_hndl,char_fine,&gl_wchar,&gl_hchar,
               &gl_wchar,gl_hchar);
    ydata++;
    startstate = 1;
    pathcount = 0;
    rate_big = FALSE;
    cnttrunc = 0;
    cntprune = 0;
    lb = 1.0;
    ub = 1.0;
    del = 0.0;

    switch (LIST) {
                case (1): break; /* NOTHING */
                case (2): hdr2(); break;
                case (3): hdr3(); break;
                } /* case */
testptr = 0L;
ltoa((LONG)testptr,num_str);
dos_printf(num_str);
ltoa((LONG)fdeath,num_str);
dos_printf(num_str);
ltoa((LONG)NULLPTR,num_str);
dos_printf(num_str);

/*   while( fdeath != NULLPTR ) */    /* Cleanup from previous runs  */
/*  {
        d = fdeath;
        fdeath = fdeath->next;
dos_printf("The pointer for d,DEATREC, to be freed follows");
ltoa((LONG)d,num_str);
dos_printf(num_str);
        free(d);
    } */
    fdeath = NULLPTR;
```

```
traverse(startstate,NULL,NULL,lb,ub,del);

d = fdeath;
lbfail = 0.0;
ubfail = 0.0;
if( LIST >= 3 )
     hdr2();
while( d != NULLPTR )
{
     if( LIST >= 2 )
     {
          itoa(d->st,num_str);
          v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 num_str);
          fota((d->lprob),lprobstr);
          v_gtext(data_hndl,data_work.g_x+80,data_work.g_y+ydata*8,
                 lprobstr);
          fota((d->uprob),uprobstr);
          v_gtext(data_hndl,data_work.g_x+158,data_work.g_y+ydata*8,
                 uprobstr);
          ydata++;

          if( et_bad )
          {
          v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 " .. E(T) INACCURATE ");
               ydata++;
          }
          et_bad = FALSE;
     }
     lbfail = lbfail + d->lprob;
     ubfail = ubfail + d->uprob;
     d = d->next;
}
pwarn = ubfail < PRUNE*cntprune*(power(10,WARNDIS));

switch (LIST) {
     case (1):
          v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 " CASE 1 SELECTED ");
          ydata++; break;
     case (2):
     case (3):
          v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                 "TOTAL");
          fota(lbfail,lbfailstr);
          v_gtext(data_hndl,data_work.g_x+80,data_work.g_y+ydata*8,
                 lbfailstr);
          fota(ubfail,ubfailstr);
```

```
            v_gtext(data_hndl,data_work.g_x+158,data_work.g_y+ydata*8,
                    ubfailstr);
            ydata++; break;
              } /* End of Case */
    if( LIST > 0 )
    {
        if( rec_slow )
        {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    " .. RECOVERY TOO SLOW ");
                ydata++;
        }
        if( rate_big )
        {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    " .. RATE TOO FAST ");
                ydata++;
        }
        if( std_big )
        {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    " .. ST. DEV TOO BIG ");
                ydata++;
        }
        if( bigst < 0 )
        {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    " .. 0 STATES IN MODEL");
                ydata++;
        }
        if( pwarn )
        {
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    " .. PRUNING TOO SEVERE");
                ydata++;
        }
    }
    else
        erun = TRUE;
    rec_slow = FALSE;
    rate_big = FALSE;
    std_big = FALSE;

    set_clip(TRUE,&data_work);
    vst_height(data_hndl,char_fine,&gl_wchar,&gl_hchar,
              &gl_wchar,gl_hchar);
dos_printf("do_compute has been exited");
} /* do_compute() */
```

```
/*------------------------------*/
/*      execstats()             */
/*------------------------------*/
VOID
execstats()
{
    BYTE    num_str[5];
dos_printf("execstats has been entered");


    if( LIST > 0 )
    {
        itoa(pathcount,num_str);
        v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                num_str);
        v_gtext(data_hndl,data_work.g_x+83,data_work.g_y+ydata*8,
            "PATH(S) PROCESSED");
        ydata++;

        if( cnttrunc > 0 )
        {
            itoa(cnttrunc,num_str);
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    num_str);
            v_gtext(data_hndl,data_work.g_x+83,data_work.g_y+ydata*8,
                    "LOOP(S) TRUNCATED");
            ydata++;
        }
        if( cntprune > 0 )
        {
            itoa(cntprune,num_str);
            v_gtext(data_hndl,data_work.g_x+5,data_work.g_y+ydata*8,
                    num_str);
            v_gtext(data_hndl,data_work.g_x+83,data_work.g_y+ydata*8,
                    "LOOP(S) PRUNED");
            ydata++;
        }
    }
    save_data_work();
dos_printf("execstats has been exited");
} /* execstat() */
```

LIST OF REFERENCES

1. NASA Technical Memorandum 87593, The SURE Reliability Analysis Program, by Ricky W. Butler, February 1986.

2. NASA CR-172340, Upper and Lower Bounds for Semi-Markov Reliability Models of Reconfigurable Systems, by Allan L.White, 1984.

3. NASA Technical Memorandum 86261, The Semi-Markov Unreliability Range Evaluator (SURE) Program, by Ricky W. Butler, 1984.

4. NASA Technical Pamphlet 2409, Reliability Bounds for Fault-Tolerant Systems With Competing Responses To Component Failures, by Larry D. Lee, 1985.

5. NASA CR - 178008, Synthetic Bounds for Semi-Markov Reliability Models, by Allan L. White, 1985.

6. Digital Research Inc., Graphics Environment Manager (GEM) Documentation, 1985.

7. NASA CR-172146, Development and Analysis of the Software Implemented Fault-Tolerance (SIFT) Computer, by Jack Goldberg, William H. Kautz, P. Michael Melliar-Smith, W. Milton, Karl N. Levitt, N. Karl, Richard L. Schwartz, and Charles B. Weinstock, 1984.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Assistant Professor Larry W. Abbott Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	2
4.	Adjunct Professor J. T. Butler Department of Electrical and Computer Engineering Naval postgraduate School Monterey, California 93943	1
5.	Major J. C. Bordeaux 2530 Paxton St. Woodbridge, Virginia 22192	1
6.	Ricky Butler Mail Code 130 NASA Langley Research Center Hampton, Virginia 23665	1
7.	Milton Holt Mail Code 130 NASA Langley Research Center Hampton, Virginia 23665	1

An Interactive Computer Aided Design and Analysis Package

by

John C. Bordeaux
Major, United States Marine Corps
B.S., Old Dominion University, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1986

Author: John C. Bordeaux
John C. Bordeaux

Approved by: L. W. Abbott
L. W. Abbott, Thesis Advisor

J. T. Butler
J. T. Butler, Second Reader,
Department of Electrical and Computer Engineering

Harriett Rigas
Harriett B. Rigas, Chairman,
Department of Electrical and Computer Engineering

J N Dyer
John Dyer, Dean of Science and Engineering

ABSTRACT

This thesis describes the porting of a NASA developed Markov reliability analysis tool to a relatively inexpensive IBM-AT. Currently the Markov analysis tool, called SURE, is not widely utilized because it runs in an expensive environment consisting of a VAX, megatex display, and template graphics software. Although substantial savings can be made by running SURE without the expensive graphics, the user friendliness of the tool is dramatically degraded by the lack of graphics. Accordingly a C program using the inexpensive GEM graphics environment has been written. The program is user friendly; it uses menus for option selections and prompts for data entry.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Assistant Professor Larry W. Abbott Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	2
4.	Adjunct Professor J. T. Butler Department of Electrical and Computer Engineering Naval postgraduate School Monterey, California 93943	1
5.	Major J. C. Bordeaux 2530 Paxton St. Woodbridge, Virginia 22192	1
6.	Ricky Butler Mail Code 130 NASA Langley Research Center Hampton, Virginia 23665	1
7.	Milton Holt Mail Code 130 NASA Langley Research Center Hampton, Virginia 23665	1